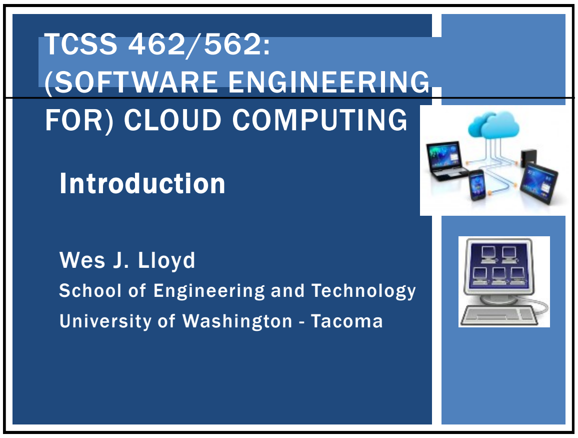
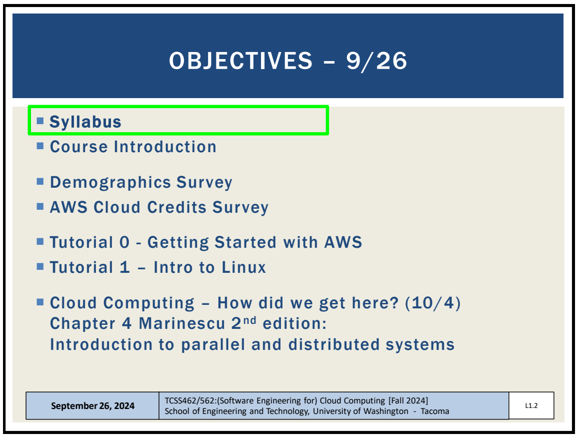


TCSS 462/562: (SOFTWARE ENGINEERING FOR) CLOUD COMPUTING

Introduction

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

1

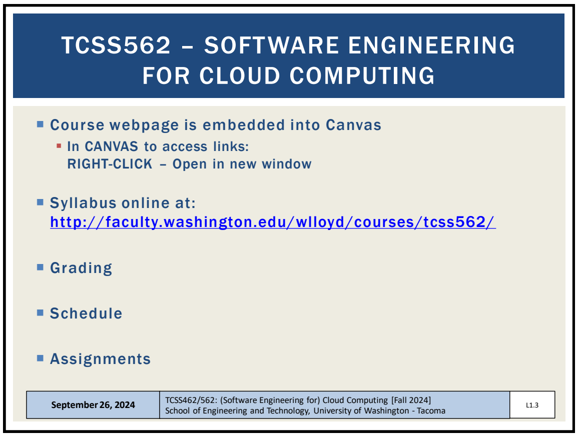


OBJECTIVES – 9/26

- Syllabus
- Course Introduction
- Demographics Survey
- AWS Cloud Credits Survey
- Tutorial 0 - Getting Started with AWS
- Tutorial 1 – Intro to Linux
- Cloud Computing – How did we get here? (10/4)
  Chapter 4 Marinescu 2nd edition:
  Introduction to parallel and distributed systems

September 26, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.2

2

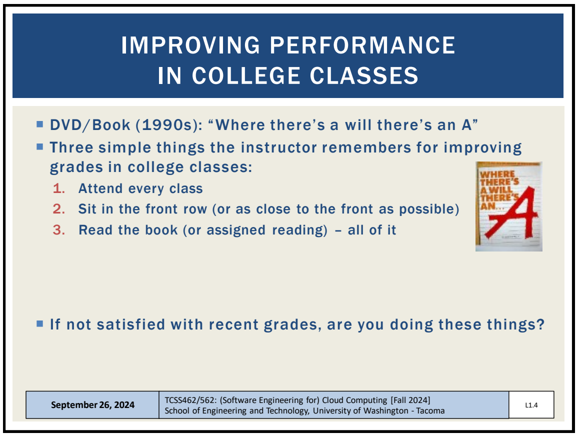


TCSS562 – SOFTWARE ENGINEERING FOR CLOUD COMPUTING

- Course webpage is embedded into Canvas
  - In CANVAS to access links:
    RIGHT-CLICK – Open in new window
- Syllabus online at:
  http://faculty.washington.edu/wlloyd/courses/tcss562/
- Grading
- Schedule
- Assignments

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.3

3

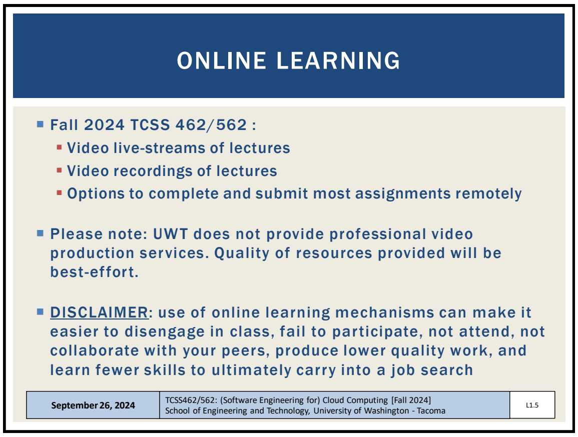


IMPROVING PERFORMANCE IN COLLEGE CLASSES

- DVD/Book (1990s): “Where there’s a will there’s an A”
- Three simple things the instructor remembers for improving grades in college classes:
  1. Attend every class
  2. Sit in the front row (or as close to the front as possible)
  3. Read the book (or assigned reading) – all of it
- If not satisfied with recent grades, are you doing these things?

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.4

4

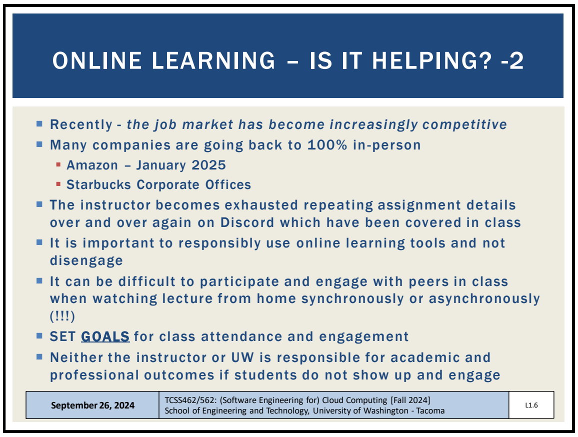


ONLINE LEARNING

- Fall 2024 TCSS 462/562 :
  - Video live-streams of lectures
  - Video recordings of lectures
  - Options to complete and submit most assignments remotely
- Please note: UWT does not provide professional video production services. Quality of resources provided will be best-effort.
- DISCLAIMER: use of online learning mechanisms can make it easier to disengage in class, fail to participate, not attend, not collaborate with your peers, produce lower quality work, and learn fewer skills to ultimately carry into a job search

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.5

5

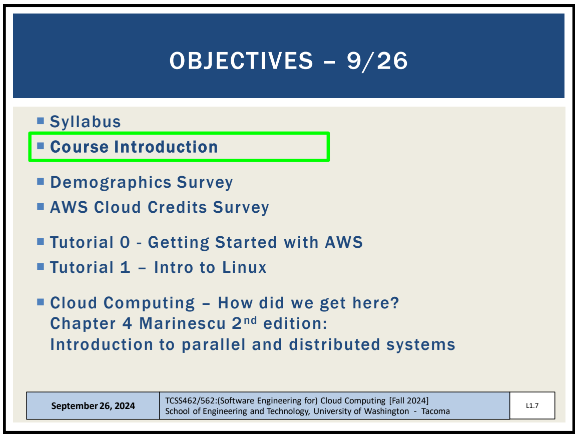


ONLINE LEARNING – IS IT HELPING? -2

- Recently - *the job market has become increasingly competitive*
- Many companies are going back to 100% in-person
  - Amazon – January 2025
  - Starbucks Corporate Offices
- The instructor becomes exhausted repeating assignment details over and over again on Discord which have been covered in class
- It is important to responsibly use online learning tools and not disengage
- It can be difficult to participate and engage with peers in class when watching lecture from home synchronously or asynchronously (!!!)
- SET **GOALS** for class attendance and engagement
- Neither the instructor or UW is responsible for academic and professional outcomes if students do not show up and engage

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.6

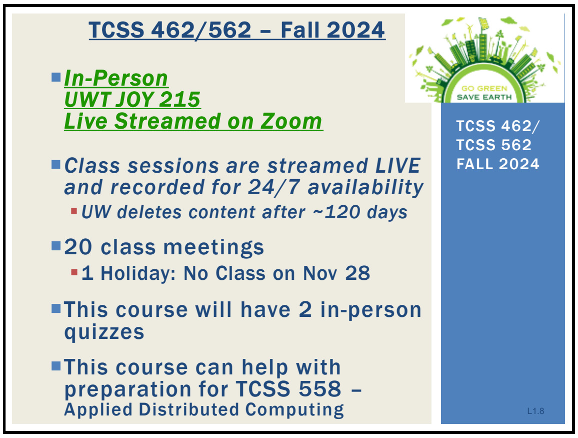
6



OBJECTIVES – 9/26

- Syllabus
- Course Introduction
- Demographics Survey
- AWS Cloud Credits Survey
- Tutorial 0 - Getting Started with AWS
- Tutorial 1 – Intro to Linux
- Cloud Computing – How did we get here?
Chapter 4 Marinescu 2nd edition:
Introduction to parallel and distributed systems

7



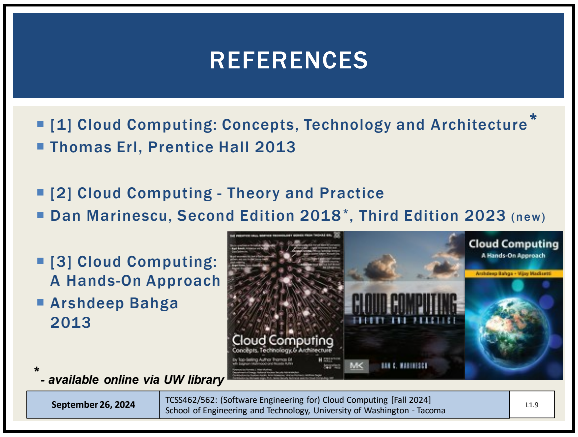
TCSS 462/562 – Fall 2024

- *In-Person*
UWT JOY 215
Live Streamed on Zoom
- *Class sessions are streamed LIVE and recorded for 24/7 availability*
  - *UW deletes content after ~120 days*
- 20 class meetings
  - 1 Holiday: No Class on Nov 28
- This course will have 2 in-person quizzes
- This course can help with preparation for TCSS 558 – Applied Distributed Computing

GO GREEN
SAVE EARTH

TCSS 462/
TCSS 562
FALL 2024

8



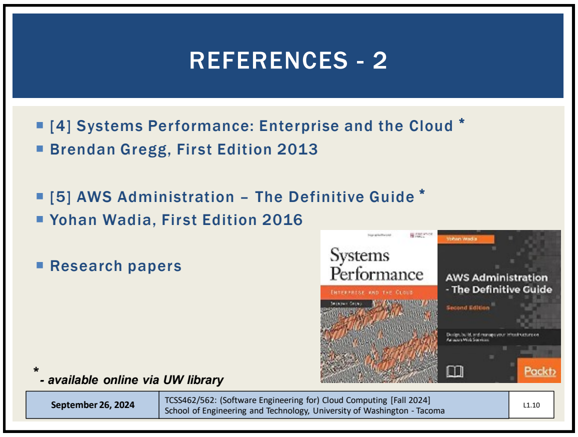
REFERENCES

- [1] Cloud Computing: Concepts, Technology and Architecture*
- Thomas Erl, Prentice Hall 2013
- [2] Cloud Computing - Theory and Practice
- Dan Marinescu, Second Edition 2018*, Third Edition 2023 (new)
- [3] Cloud Computing: A Hands-On Approach
- Arshdeep Bahga 2013

Cloud Computing
Concepts, Technology & Architecture
CLOUD COMPUTING
THEORY AND PRACTICE
DAN C. MARINESCU
Cloud Computing
A Hands-On Approach
Arshdeep Bahga • Vijay Madisetti

* - available online via UW library

9



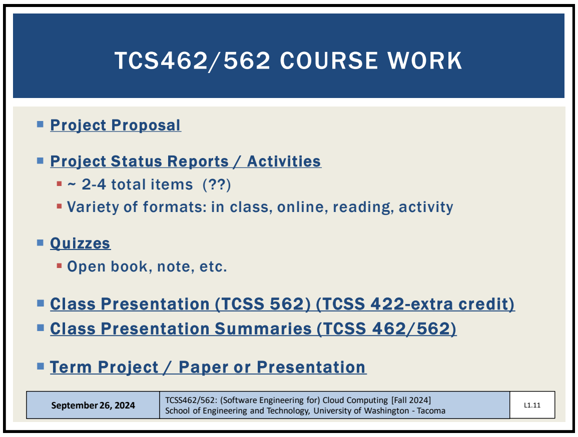
REFERENCES - 2

- [4] Systems Performance: Enterprise and the Cloud *
- Brendan Gregg, First Edition 2013
- [5] AWS Administration – The Definitive Guide *
- Yohan Wadia, First Edition 2016
- Research papers

Systems
Performance
ENTERPRISE AND THE CLOUD
Yohan Wadia
AWS Administration
- The Definitive Guide
Second Edition
Packt

* - available online via UW library

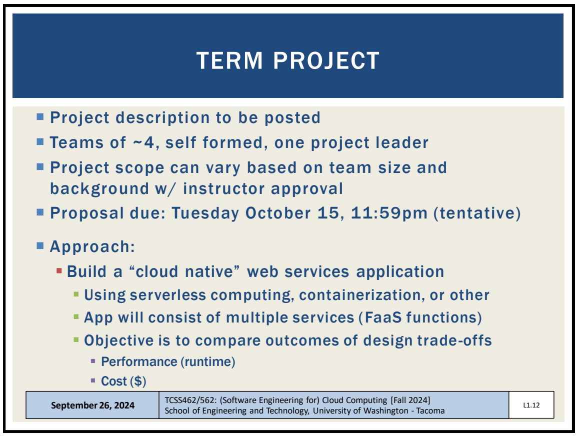
10



TCS462/562 COURSE WORK

- Project Proposal
- Project Status Reports / Activities
  - ~ 2-4 total items (??)
  - Variety of formats: in class, online, reading, activity
- Quizzes
  - Open book, note, etc.
- Class Presentation (TCSS 562) (TCSS 422-extra credit)
- Class Presentation Summaries (TCSS 462/562)
- Term Project / Paper or Presentation

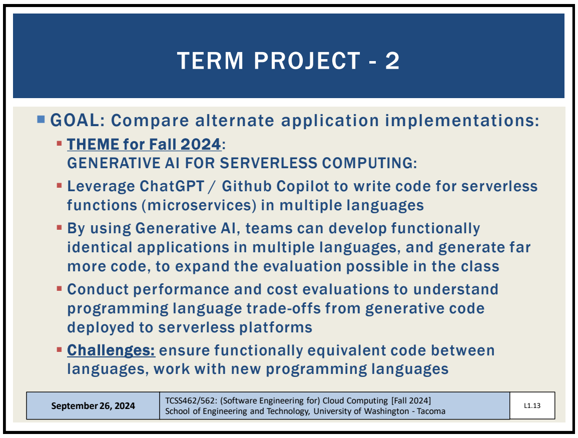
11



TERM PROJECT

- Project description to be posted
- Teams of ~4, self formed, one project leader
- Project scope can vary based on team size and background w/ instructor approval
- Proposal due: Tuesday October 15, 11:59pm (tentative)
- Approach:
  - Build a “cloud native” web services application
    - Using serverless computing, containerization, or other
    - App will consist of multiple services (FaaS functions)
    - Objective is to compare outcomes of design trade-offs
      - Performance (runtime)
      - Cost ($)

12

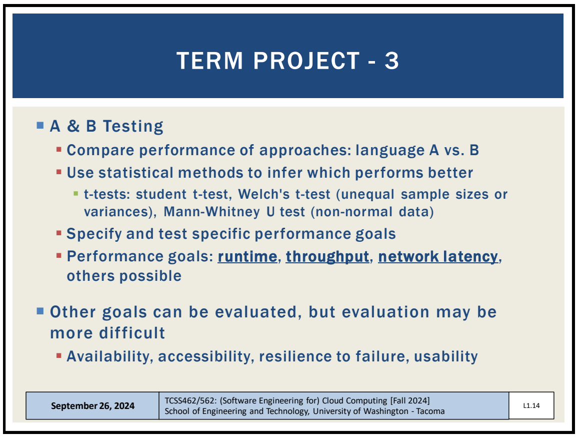


TERM PROJECT - 2

- GOAL: Compare alternate application implementations:
  - **THEME for Fall 2024**:
    GENERATIVE AI FOR SERVERLESS COMPUTING:
  - Leverage ChatGPT / Github Copilot to write code for serverless functions (microservices) in multiple languages
  - By using Generative AI, teams can develop functionally identical applications in multiple languages, and generate far more code, to expand the evaluation possible in the class
  - Conduct performance and cost evaluations to understand programming language trade-offs from generative code deployed to serverless platforms
  - **Challenges:** ensure functionally equivalent code between languages, work with new programming languages

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.13

13

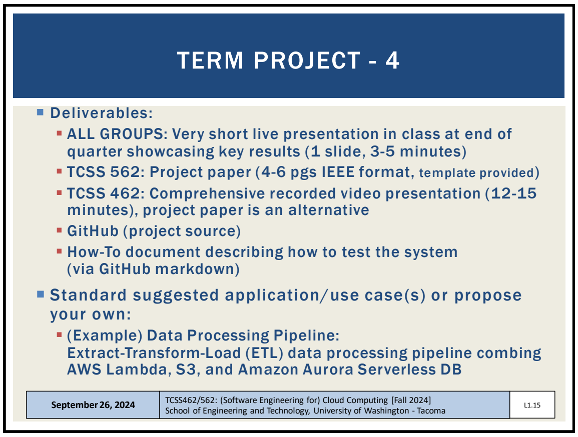


TERM PROJECT - 3

- A & B Testing
  - Compare performance of approaches: language A vs. B
  - Use statistical methods to infer which performs better
    - t-tests: student t-test, Welch's t-test (unequal sample sizes or variances), Mann-Whitney U test (non-normal data)
  - Specify and test specific performance goals
  - Performance goals: **runtime**, **throughput**, **network latency**, others possible
- Other goals can be evaluated, but evaluation may be more difficult
  - Availability, accessibility, resilience to failure, usability

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.14

14

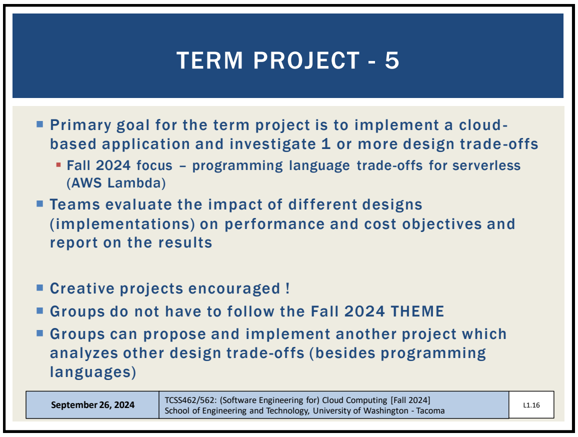


TERM PROJECT - 4

- Deliverables:
  - ALL GROUPS: Very short live presentation in class at end of quarter showcasing key results (1 slide, 3-5 minutes)
  - TCSS 562: Project paper (4-6 pgs IEEE format, template provided)
  - TCSS 462: Comprehensive recorded video presentation (12-15 minutes), project paper is an alternative
  - GitHub (project source)
  - How-To document describing how to test the system (via GitHub markdown)
- Standard suggested application/use case(s) or propose your own:
  - (Example) Data Processing Pipeline: Extract-Transform-Load (ETL) data processing pipeline combing AWS Lambda, S3, and Amazon Aurora Serverless DB

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.15

15

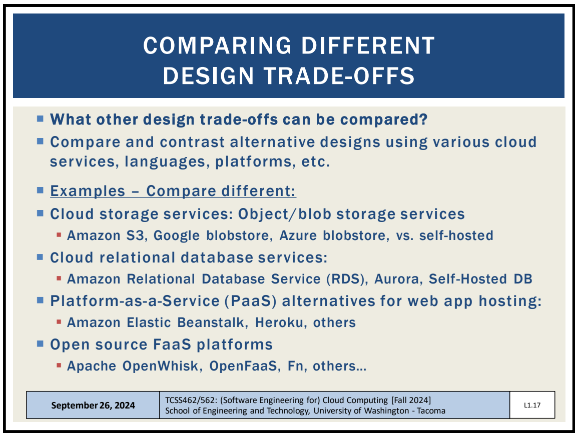


TERM PROJECT - 5

- Primary goal for the term project is to implement a cloud-based application and investigate 1 or more design trade-offs
  - Fall 2024 focus – programming language trade-offs for serverless (AWS Lambda)
- Teams evaluate the impact of different designs (implementations) on performance and cost objectives and report on the results
- Creative projects encouraged !
- Groups do not have to follow the Fall 2024 THEME
- Groups can propose and implement another project which analyzes other design trade-offs (besides programming languages)

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.16

16

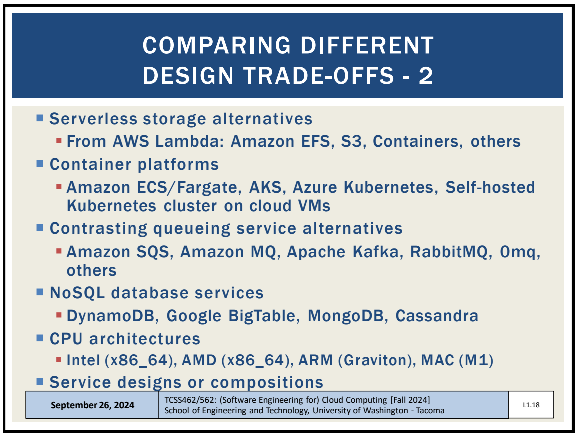


COMPARING DIFFERENT DESIGN TRADE-OFFS

- **What other design trade-offs can be compared?**
- Compare and contrast alternative designs using various cloud services, languages, platforms, etc.
- Examples – Compare different:
- Cloud storage services: Object/blob storage services
  - Amazon S3, Google blobstore, Azure blobstore, vs. self-hosted
- Cloud relational database services:
  - Amazon Relational Database Service (RDS), Aurora, Self-Hosted DB
- Platform-as-a-Service (PaaS) alternatives for web app hosting:
  - Amazon Elastic Beanstalk, Heroku, others
- Open source FaaS platforms
  - Apache OpenWhisk, OpenFaaS, Fn, others…

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.17

17

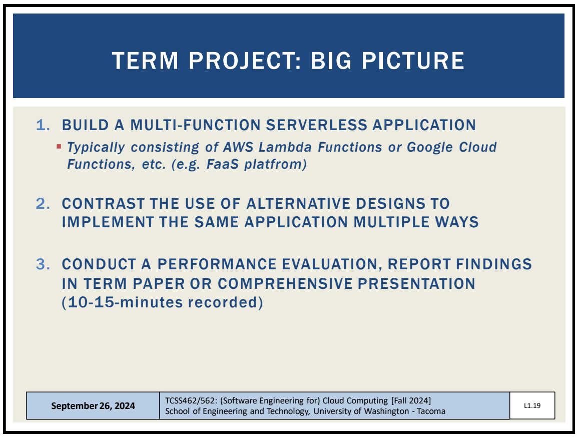


COMPARING DIFFERENT DESIGN TRADE-OFFS - 2

- Serverless storage alternatives
  - From AWS Lambda: Amazon EFS, S3, Containers, others
- Container platforms
  - Amazon ECS/Fargate, AKS, Azure Kubernetes, Self-hosted Kubernetes cluster on cloud VMs
- Contrasting queueing service alternatives
  - Amazon SQS, Amazon MQ, Apache Kafka, RabbitMQ, Omq, others
- NoSQL database services
  - DynamoDB, Google BigTable, MongoDB, Cassandra
- CPU architectures
  - Intel (x86_64), AMD (x86_64), ARM (Graviton), MAC (M1)
- Service designs or compositions

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.18

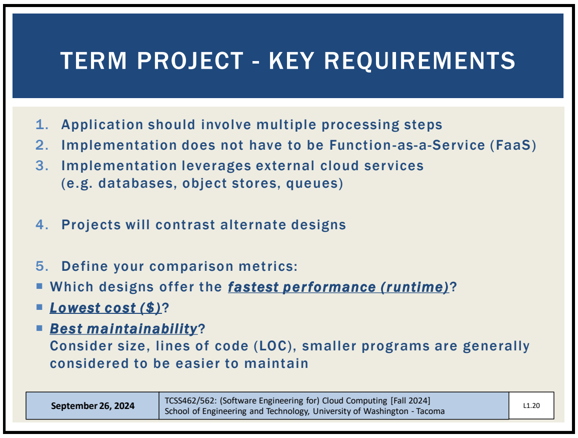
18



TERM PROJECT: BIG PICTURE

1. BUILD A MULTI-FUNCTION SERVERLESS APPLICATION
   - *Typically consisting of AWS Lambda Functions or Google Cloud Functions, etc. (e.g. FaaS platfrom)*
2. CONTRAST THE USE OF ALTERNATIVE DESIGNS TO IMPLEMENT THE SAME APPLICATION MULTIPLE WAYS
3. CONDUCT A PERFORMANCE EVALUATION, REPORT FINDINGS IN TERM PAPER OR COMPREHENSIVE PRESENTATION (10-15-minutes recorded)

19

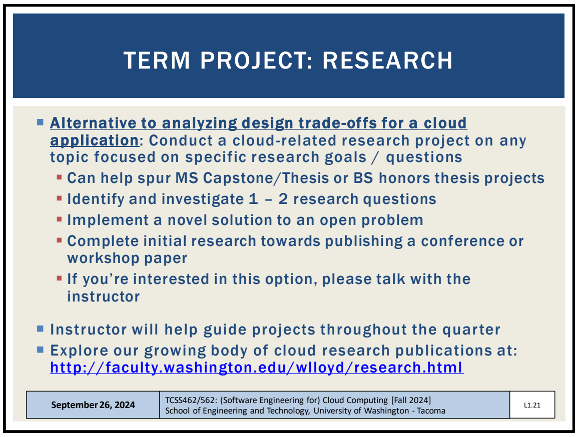


TERM PROJECT - KEY REQUIREMENTS

1. Application should involve multiple processing steps
2. Implementation does not have to be Function-as-a-Service (FaaS)
3. Implementation leverages external cloud services (e.g. databases, object stores, queues)
4. Projects will contrast alternate designs
5. Define your comparison metrics:

- Which designs offer the ***fastest performance (runtime)***?
- ***Lowest cost ($)***?
- ***Best maintainability***?
  Consider size, lines of code (LOC), smaller programs are generally considered to be easier to maintain

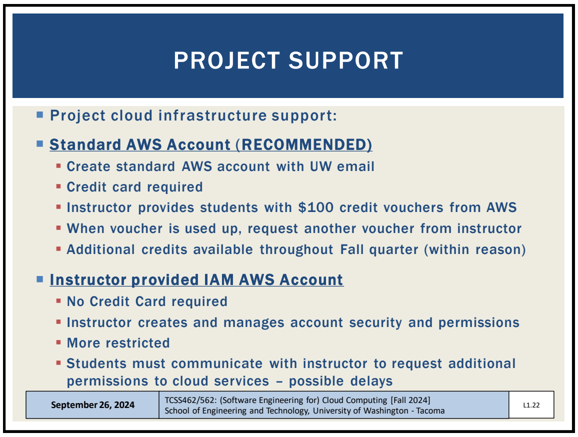
20



TERM PROJECT: RESEARCH

- **Alternative to analyzing design trade-offs for a cloud application**: Conduct a cloud-related research project on any topic focused on specific research goals / questions
  - Can help spur MS Capstone/Thesis or BS honors thesis projects
  - Identify and investigate 1 – 2 research questions
  - Implement a novel solution to an open problem
  - Complete initial research towards publishing a conference or workshop paper
  - If you're interested in this option, please talk with the instructor
- Instructor will help guide projects throughout the quarter
- Explore our growing body of cloud research publications at: http://faculty.washington.edu/wlloyd/research.html

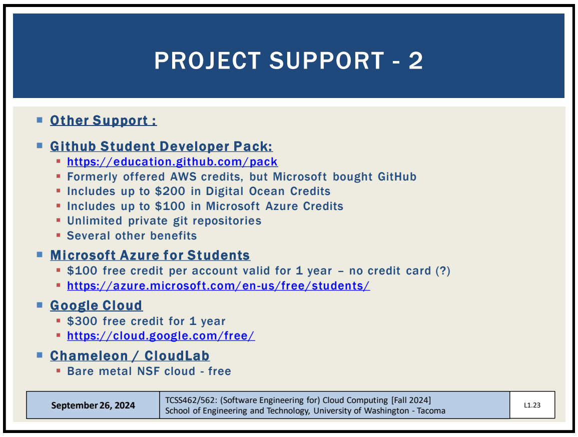
21



PROJECT SUPPORT

- Project cloud infrastructure support:
- **Standard AWS Account (RECOMMENDED)**
  - Create standard AWS account with UW email
  - Credit card required
  - Instructor provides students with $100 credit vouchers from AWS
  - When voucher is used up, request another voucher from instructor
  - Additional credits available throughout Fall quarter (within reason)
- **Instructor provided IAM AWS Account**
  - No Credit Card required
  - Instructor creates and manages account security and permissions
  - More restricted
  - Students must communicate with instructor to request additional permissions to cloud services – possible delays

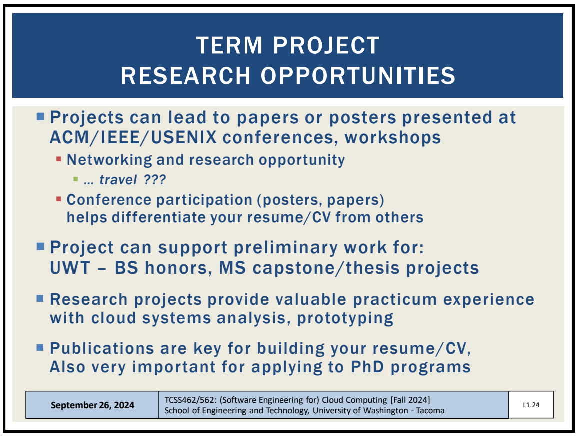
22



PROJECT SUPPORT - 2

- **Other Support :**
- **Github Student Developer Pack:**
  - https://education.github.com/pack
  - Formerly offered AWS credits, but Microsoft bought GitHub
  - Includes up to $200 in Digital Ocean Credits
  - Includes up to $100 in Microsoft Azure Credits
  - Unlimited private git repositories
  - Several other benefits
- **Microsoft Azure for Students**
  - $100 free credit per account valid for 1 year – no credit card (?)
  - https://azure.microsoft.com/en-us/free/students/
- **Google Cloud**
  - $300 free credit for 1 year
  - https://cloud.google.com/free/
- **Chameleon / CloudLab**
  - Bare metal NSF cloud - free

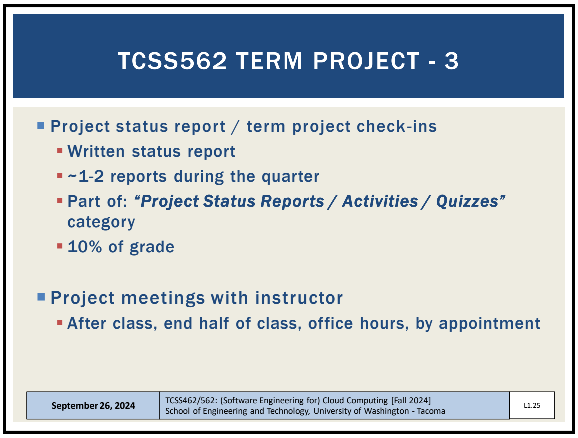
23



TERM PROJECT RESEARCH OPPORTUNITIES

- Projects can lead to papers or posters presented at ACM/IEEE/USENIX conferences, workshops
  - Networking and research opportunity
    - *… travel ???*
  - Conference participation (posters, papers) helps differentiate your resume/CV from others
- Project can support preliminary work for: UWT – BS honors, MS capstone/thesis projects
- Research projects provide valuable practicum experience with cloud systems analysis, prototyping
- Publications are key for building your resume/CV, Also very important for applying to PhD programs

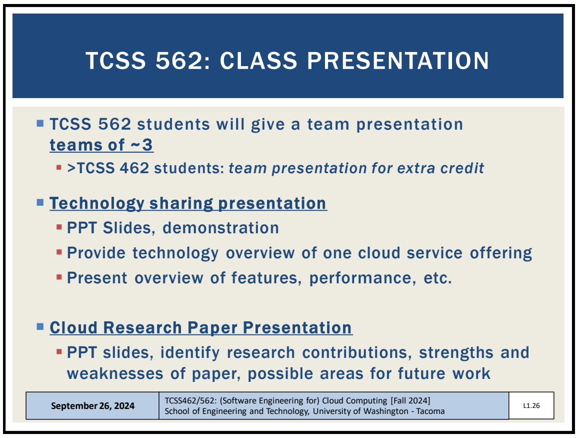
24



TCSS562 TERM PROJECT - 3

- Project status report / term project check-ins
  - Written status report
  - ~1-2 reports during the quarter
  - Part of: ***"Project Status Reports / Activities / Quizzes"*** category
  - 10% of grade
- Project meetings with instructor
  - After class, end half of class, office hours, by appointment

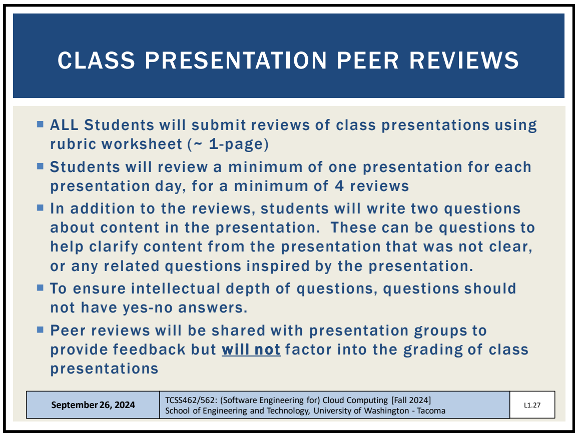
25



TCSS 562: CLASS PRESENTATION

- TCSS 562 students will give a team presentation **teams of ~3**
  - >TCSS 462 students: *team presentation for extra credit*
- **Technology sharing presentation**
  - PPT Slides, demonstration
  - Provide technology overview of one cloud service offering
  - Present overview of features, performance, etc.
- **Cloud Research Paper Presentation**
  - PPT slides, identify research contributions, strengths and weaknesses of paper, possible areas for future work

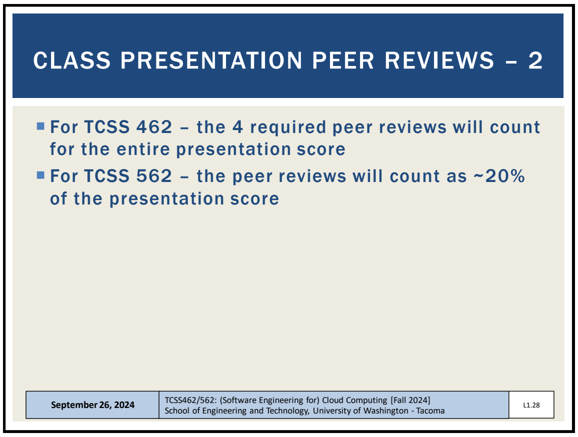
26



CLASS PRESENTATION PEER REVIEWS

- ALL Students will submit reviews of class presentations using rubric worksheet (~ 1-page)
- Students will review a minimum of one presentation for each presentation day, for a minimum of 4 reviews
- In addition to the reviews, students will write two questions about content in the presentation. These can be questions to help clarify content from the presentation that was not clear, or any related questions inspired by the presentation.
- To ensure intellectual depth of questions, questions should not have yes-no answers.
- Peer reviews will be shared with presentation groups to provide feedback but **will not** factor into the grading of class presentations

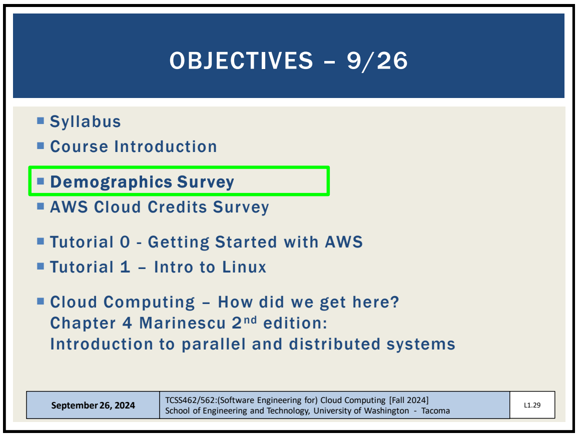
27



CLASS PRESENTATION PEER REVIEWS – 2

- For TCSS 462 – the 4 required peer reviews will count for the entire presentation score
- For TCSS 562 – the peer reviews will count as ~20% of the presentation score

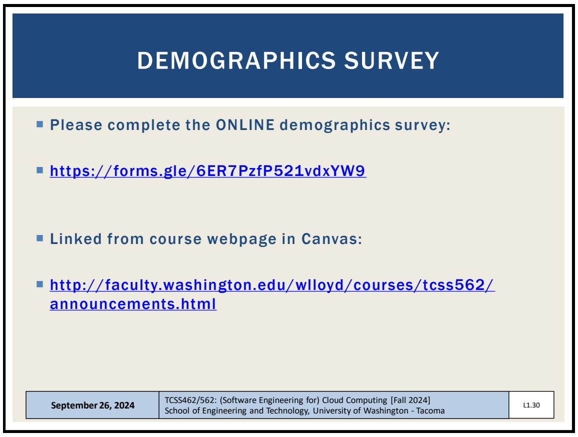
28



OBJECTIVES – 9/26

- Syllabus
- Course Introduction
- **Demographics Survey**
- AWS Cloud Credits Survey
- Tutorial 0 - Getting Started with AWS
- Tutorial 1 – Intro to Linux
- Cloud Computing – How did we get here?
  Chapter 4 Marinescu 2nd edition:
  Introduction to parallel and distributed systems

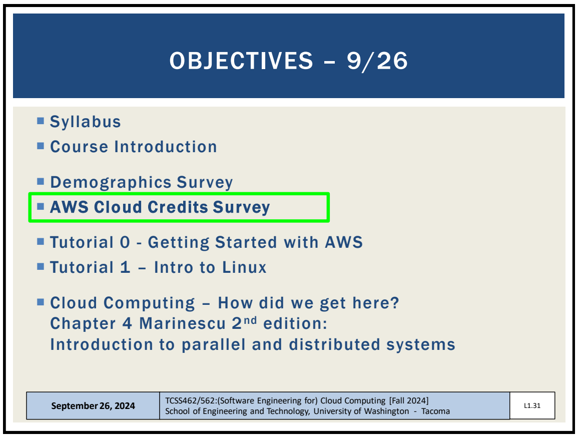
29



DEMOGRAPHICS SURVEY

- Please complete the ONLINE demographics survey:
- https://forms.gle/6ER7PzfP521vdxYW9
- Linked from course webpage in Canvas:
- http://faculty.washington.edu/wlloyd/courses/tcss562/announcements.html

30

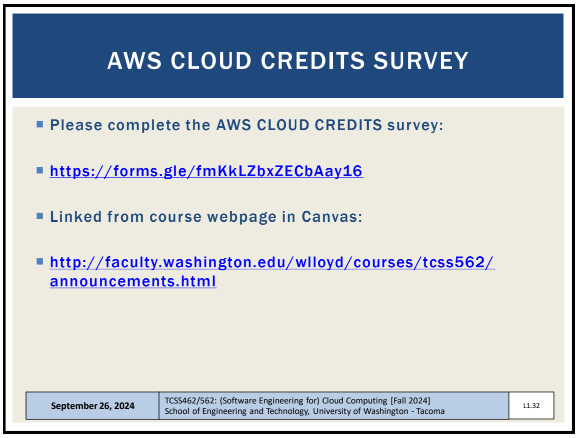


OBJECTIVES – 9/26

- Syllabus
- Course Introduction
- Demographics Survey
- **AWS Cloud Credits Survey**
- Tutorial 0 - Getting Started with AWS
- Tutorial 1 – Intro to Linux
- Cloud Computing – How did we get here?
  Chapter 4 Marinescu 2nd edition:
  Introduction to parallel and distributed systems

September 26, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.31

31

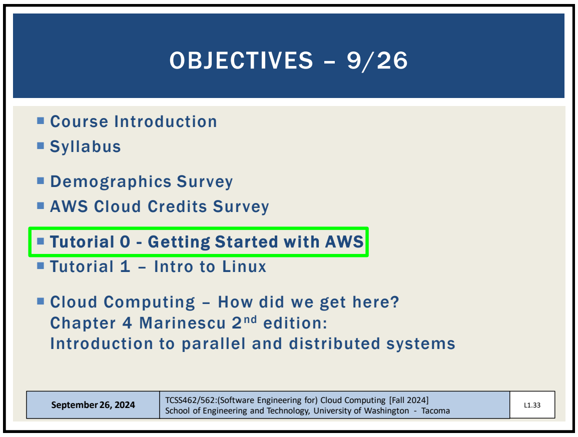


AWS CLOUD CREDITS SURVEY

- Please complete the AWS CLOUD CREDITS survey:
- https://forms.gle/fmKkLZbxZECbAay16
- Linked from course webpage in Canvas:
- http://faculty.washington.edu/wlloyd/courses/tcss562/announcements.html

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.32

32

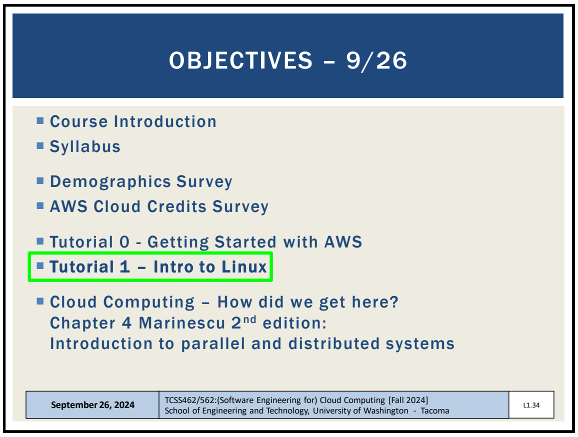


OBJECTIVES – 9/26

- Course Introduction
- Syllabus
- Demographics Survey
- AWS Cloud Credits Survey
- **Tutorial 0 - Getting Started with AWS**
- Tutorial 1 – Intro to Linux
- Cloud Computing – How did we get here?
  Chapter 4 Marinescu 2nd edition:
  Introduction to parallel and distributed systems

September 26, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.33

33

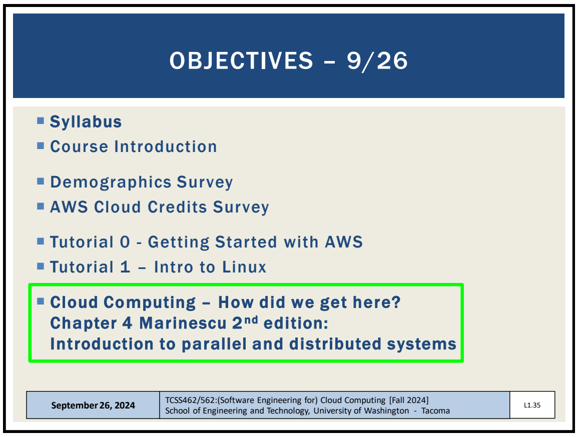


OBJECTIVES – 9/26

- Course Introduction
- Syllabus
- Demographics Survey
- AWS Cloud Credits Survey
- Tutorial 0 - Getting Started with AWS
- **Tutorial 1 – Intro to Linux**
- Cloud Computing – How did we get here?
  Chapter 4 Marinescu 2nd edition:
  Introduction to parallel and distributed systems

September 26, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.34

34

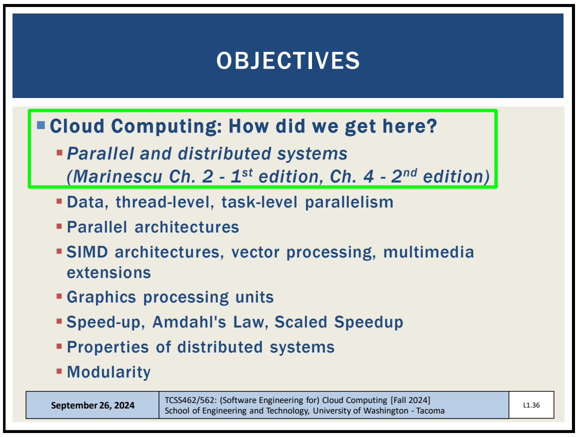


OBJECTIVES – 9/26

- Syllabus
- Course Introduction
- Demographics Survey
- AWS Cloud Credits Survey
- Tutorial 0 - Getting Started with AWS
- Tutorial 1 – Intro to Linux
- **Cloud Computing – How did we get here?
  Chapter 4 Marinescu 2nd edition:
  Introduction to parallel and distributed systems**

September 26, 2024 | TCSS462/562:(Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.35

35



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

September 26, 2024 | TCSS462/562: (Software Engineering for) Cloud Computing [Fall 2024] School of Engineering and Technology, University of Washington - Tacoma | L1.36

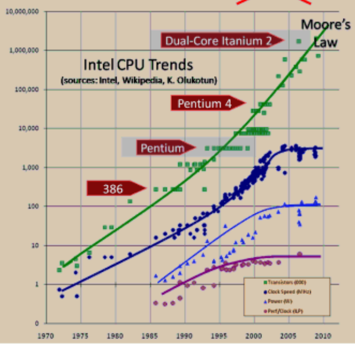
36

CLOUD COMPUTING: HOW DID WE GET HERE?

- General interest in parallel computing
  - Moore's Law - # of transistors doubles every 18 months
  - Post 2004: heat dissipation challenges: can no longer easily increase cloud speed
  - Overclocking to 7GHz takes more than just liquid nitrogen:
    - https://tinyurl.com/y93s2yz2

- Solutions:
  - Vary CPU clock speed
  - Add CPU cores
  - Multi-core technology

Each Year We Get ~~Faster~~ More Processors



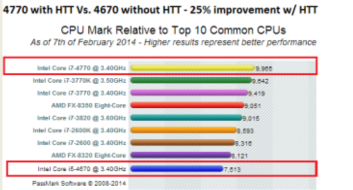
Historically: Boost single-stream performance via more complex chips.

Now: Deliver more cores per chip (+ GPU, NIC, SoC).

The free lunch is over for today's sequential apps and many concurrent apps. We need killer apps with lots of latent parallelism.

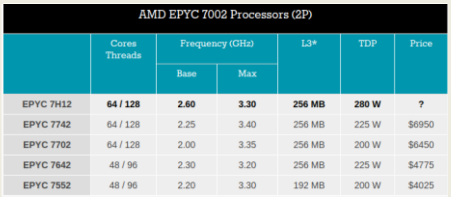
HYPER THREADING

- Modern CPUs provide multiple instruction pipelines, supporting multiple execution threads, usually 2 to feed instructions to a single CPU core…
- Two hyper-threads are not equivalent to (2) CPU cores
- i7-4770 and i5-4760 same CPU, with and without HTT
- Example: → hyperthreads add +32.9%



AMD'S 64-CORE <14NM CPUS

- Epyc Rome CPUs
- Announced August 2019
- EPYC 7H12 requires liquid cooling



AMD EPYC 7002 Processors (2P)

	Cores Threads	Frequency (GHz)		L3*	TDP	Price
		Base	Max			
EPYC 7H12	**64 / 128**	**2.60**	**3.30**	**256 MB**	**280 W**	**?**
EPYC 7742	64 / 128	2.25	3.40	256 MB	225 W	$6950
EPYC 7702	64 / 128	2.00	3.35	256 MB	200 W	$6450
EPYC 7642	48 / 96	2.30	3.20	256 MB	225 W	$4775
EPYC 7552	48 / 96	2.20	3.30	192 MB	200 W	$4025

AMD'S 64-CORE <14NM CPUS

AMD EPYC 9654/9654P/9684X/9R14 (AWS OEM):

- June 2023: ***96 cores***, 192 hyper-threads CPUs
- Mixes 4nm:APU (combines CPUs+GPU), 5nm:L3 cache (8 CPU-chiplet), and 6nm:I/O dies, 2.25 to 3.7 burst GHz, up to 400 watts
- $10,625 to $14,756

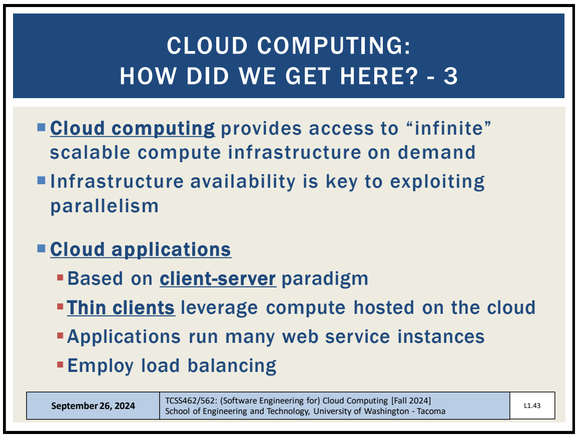
AMD EPYC 9754: ***128 cores***, 256 hyperthreads !

- 2.25 to 3.1 burst GHz, 360 watts
- $11,900

AMD EPYC 9005: ***192 cores***, 384 threads, 3nm (in dev)

CLOUD COMPUTING: HOW DID WE GET HERE? - 2

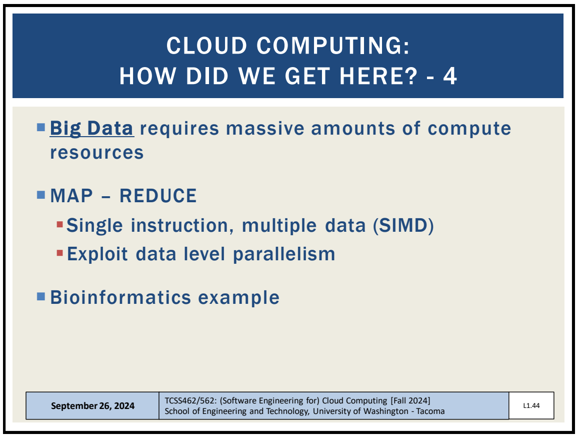
- To make computing faster, we must go "parallel"
- Difficult to expose parallelism in scientific applications
- Not every problem solution has a parallel algorithm
  - Chicken and egg problem…
- Many commercial efforts promoting pure parallel programming efforts have failed
- Enterprise computing world has been *skeptical* and less involved in parallel programming



CLOUD COMPUTING: HOW DID WE GET HERE? - 3

- **Cloud computing** provides access to "infinite" scalable compute infrastructure on demand
- Infrastructure availability is key to exploiting parallelism
- **Cloud applications**
  - Based on **client-server** paradigm
  - **Thin clients** leverage compute hosted on the cloud
  - Applications run many web service instances
  - Employ load balancing

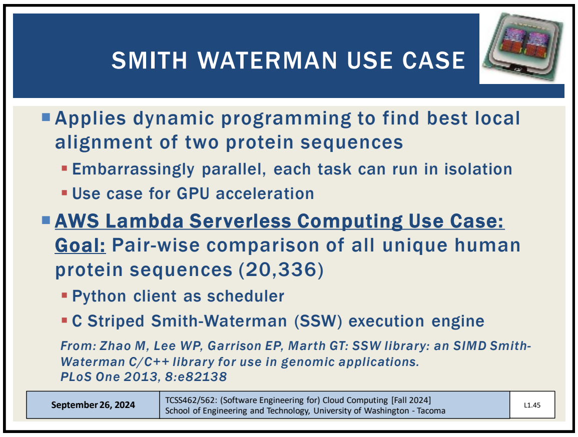
43



CLOUD COMPUTING: HOW DID WE GET HERE? - 4

- **Big Data** requires massive amounts of compute resources
- MAP – REDUCE
  - Single instruction, multiple data (SIMD)
  - Exploit data level parallelism
- Bioinformatics example

44

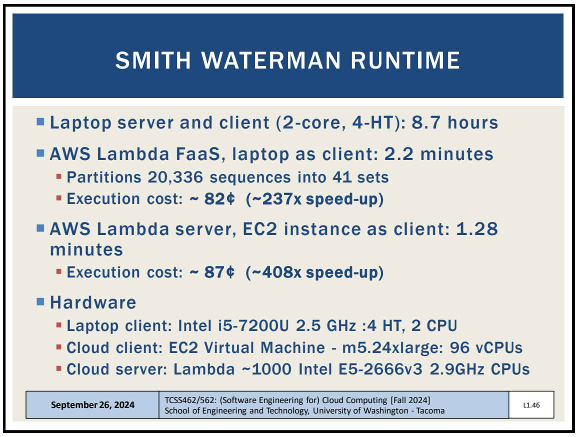


SMITH WATERMAN USE CASE

- Applies dynamic programming to find best local alignment of two protein sequences
  - Embarrassingly parallel, each task can run in isolation
  - Use case for GPU acceleration
- **AWS Lambda Serverless Computing Use Case:** **Goal:** Pair-wise comparison of all unique human protein sequences (20,336)
  - Python client as scheduler
  - C Striped Smith-Waterman (SSW) execution engine

From: Zhao M, Lee WP, Garrison EP, Marth GT: SSW library: an SIMD Smith-Waterman C/C++ library for use in genomic applications. PLoS One 2013, 8:e82138

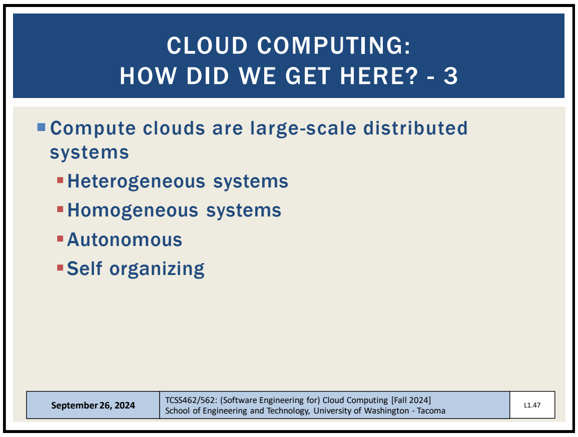
45



SMITH WATERMAN RUNTIME

- Laptop server and client (2-core, 4-HT): 8.7 hours
- AWS Lambda FaaS, laptop as client: 2.2 minutes
  - Partitions 20,336 sequences into 41 sets
  - Execution cost: **~ 82¢ (~237x speed-up)**
- AWS Lambda server, EC2 instance as client: 1.28 minutes
  - Execution cost: **~ 87¢ (~408x speed-up)**
- Hardware
  - Laptop client: Intel i5-7200U 2.5 GHz :4 HT, 2 CPU
  - Cloud client: EC2 Virtual Machine - m5.24xlarge: 96 vCPUs
  - Cloud server: Lambda ~1000 Intel E5-2666v3 2.9GHz CPUs

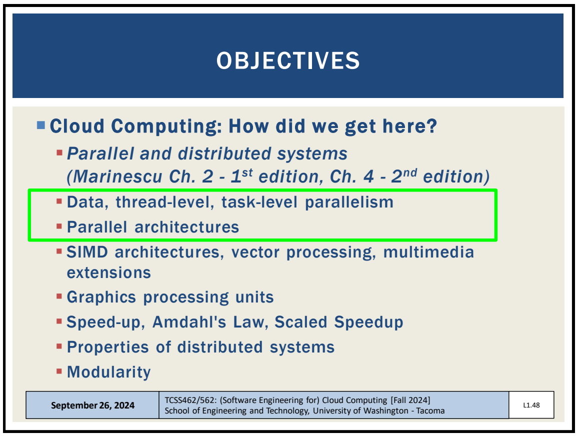
46



CLOUD COMPUTING: HOW DID WE GET HERE? - 3

- Compute clouds are large-scale distributed systems
  - Heterogeneous systems
  - Homogeneous systems
  - Autonomous
  - Self organizing

47



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

48

PARALLELISM

- Discovering parallelism and development of parallel algorithms requires considerable effort
- Example: numerical analysis problems, such as solving large systems of linear equations or solving systems of Partial Differential Equations (PDEs), require algorithms based on domain decomposition methods.
- ***How can problems be split into independent chunks?***
- **Fine-grained parallelism**
  - Only small bits of code can run in parallel without coordination
  - Communication is required to synchronize state across nodes
- **Coarse-grained parallelism**
  - Large blocks of code can run without coordination

PARALLELISM - 2

- **Coordination of nodes**
- Requires ***message passing*** or ***shared memory***
- Debugging parallel ***message passing*** code is easier than parallel ***shared memory*** code
- ***Message passing***: all of the interactions are clear
  - Coordination via specific programming API (MPI)
- ***Shared memory***: interactions can be implicit – *must read the code!!*
- Processing speed is orders of magnitude faster than communication speed (CPU > memory bus speed)
- Avoiding coordination achieves the best speed-up

TYPES OF PARALLELISM

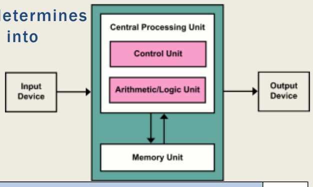
- Parallelism:
  - Goal: Perform multiple operations at the same time to achieve a speed-up
- Thread-level parallelism (TLP)
  - Control flow architecture
- Data-level parallelism
  - Data flow architecture
- Bit-level parallelism
- Instruction-level parallelism (ILP)

THREAD LEVEL PARALLELISM (TLP)

- Number of threads an application runs at any one time
- Varies throughout program execution
- As a metric:
- **Minimum**: 1 thread
- Can measure **average**, **maximum (peak)**
- **QUESTION: What are the consequences of average (TLP) for scheduling an application to run on a computer with a fixed number of CPU cores and hyperthreads?**
- Let's say there are 4 cores, or 8 hyper-threads…
- ***Key to avoiding waste of computing resources is knowing your application's TLP…***

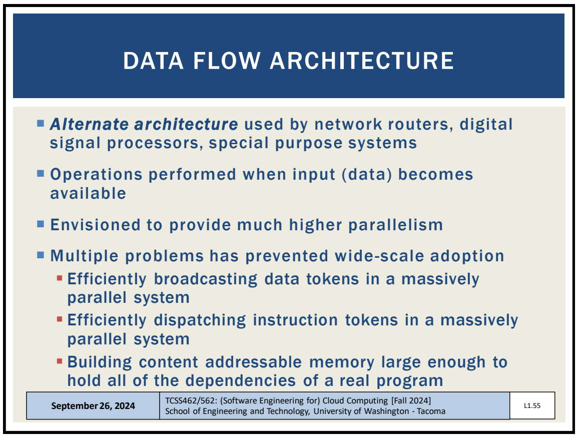
CONTROL-FLOW ARCHITECTURE

- Typical architecture used today – w/ multiple threads
- By John von Neumann (1945)
- Also called the Von Neumann architecture
- Dominant computer system architecture
- Program counter (PC) determines next instruction to load into *instruction register*
- Program execution is sequential



DATA-LEVEL PARALLELISM

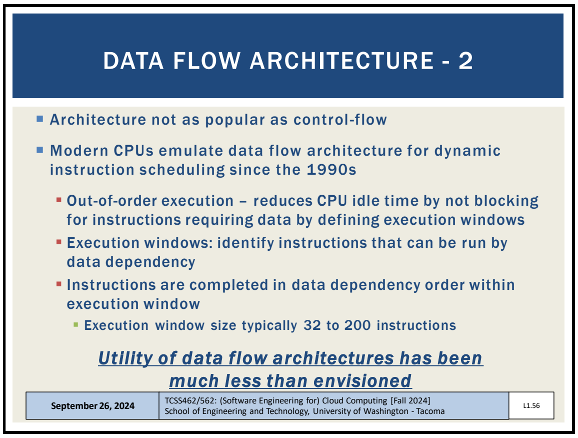
- Partition data into big chunks, run separate copies of the program on them with little or no communication
- Problems are considered to be ***embarrassingly parallel***
- Also perfectly parallel or pleasingly parallel…
- Little or no effort needed to separate problem into a number of parallel tasks
- MapReduce programming model is an example



DATA FLOW ARCHITECTURE

- ***Alternate architecture*** used by network routers, digital signal processors, special purpose systems
- Operations performed when input (data) becomes available
- Envisioned to provide much higher parallelism
- Multiple problems has prevented wide-scale adoption
  - Efficiently broadcasting data tokens in a massively parallel system
  - Efficiently dispatching instruction tokens in a massively parallel system
  - Building content addressable memory large enough to hold all of the dependencies of a real program

55

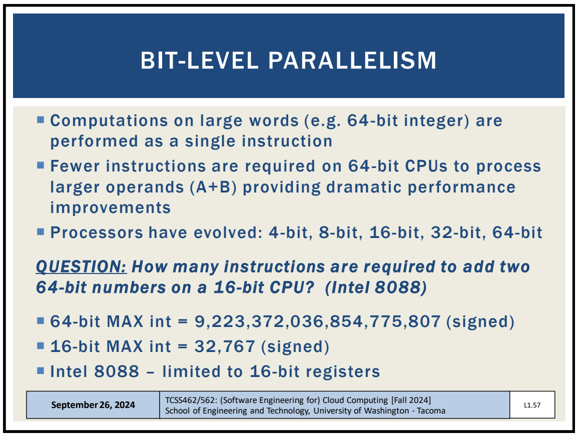


DATA FLOW ARCHITECTURE - 2

- Architecture not as popular as control-flow
- Modern CPUs emulate data flow architecture for dynamic instruction scheduling since the 1990s
  - Out-of-order execution – reduces CPU idle time by not blocking for instructions requiring data by defining execution windows
  - Execution windows: identify instructions that can be run by data dependency
  - Instructions are completed in data dependency order within execution window
    - Execution window size typically 32 to 200 instructions

Utility of data flow architectures has been much less than envisioned

56



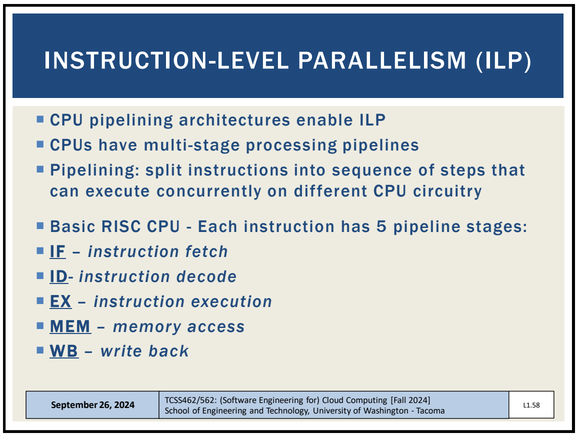
BIT-LEVEL PARALLELISM

- Computations on large words (e.g. 64-bit integer) are performed as a single instruction
- Fewer instructions are required on 64-bit CPUs to process larger operands (A+B) providing dramatic performance improvements
- Processors have evolved: 4-bit, 8-bit, 16-bit, 32-bit, 64-bit

***QUESTION:** How many instructions are required to add two 64-bit numbers on a 16-bit CPU? (Intel 8088)*

- 64-bit MAX int = 9,223,372,036,854,775,807 (signed)
- 16-bit MAX int = 32,767 (signed)
- Intel 8088 – limited to 16-bit registers

57

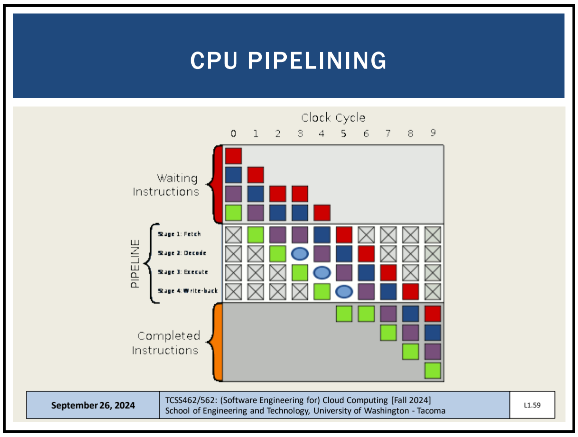


INSTRUCTION-LEVEL PARALLELISM (ILP)

- CPU pipelining architectures enable ILP
- CPUs have multi-stage processing pipelines
- Pipelining: split instructions into sequence of steps that can execute concurrently on different CPU circuitry

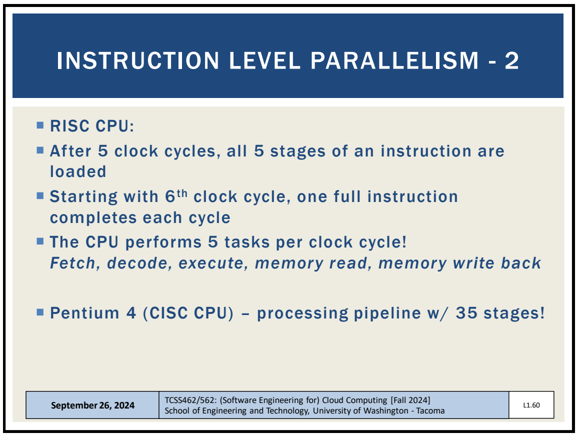
- Basic RISC CPU - Each instruction has 5 pipeline stages:
- **IF** – *instruction fetch*
- **ID**- *instruction decode*
- **EX** – *instruction execution*
- **MEM** – *memory access*
- **WB** – *write back*

58



CPU PIPELINING

59

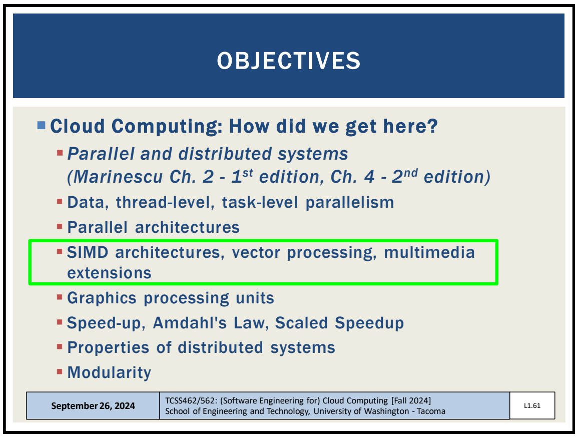


INSTRUCTION LEVEL PARALLELISM - 2

- RISC CPU:
- After 5 clock cycles, all 5 stages of an instruction are loaded
- Starting with 6th clock cycle, one full instruction completes each cycle
- The CPU performs 5 tasks per clock cycle!
  Fetch, decode, execute, memory read, memory write back

- Pentium 4 (CISC CPU) – processing pipeline w/ 35 stages!

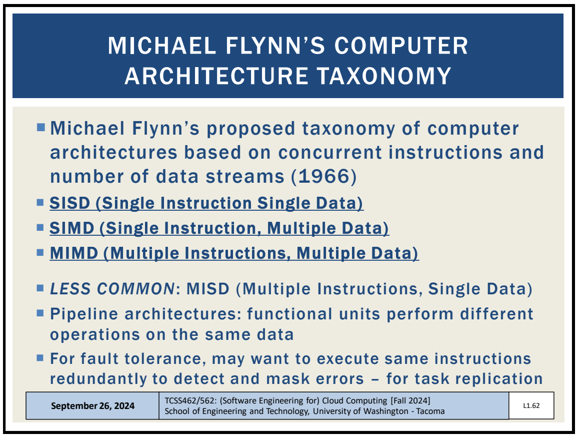
60



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

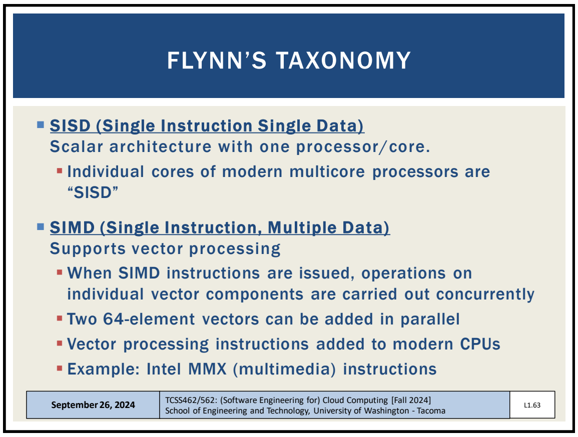
61



MICHAEL FLYNN'S COMPUTER ARCHITECTURE TAXONOMY

- Michael Flynn's proposed taxonomy of computer architectures based on concurrent instructions and number of data streams (1966)
- SISD (Single Instruction Single Data)
- SIMD (Single Instruction, Multiple Data)
- MIMD (Multiple Instructions, Multiple Data)
- *LESS COMMON*: MISD (Multiple Instructions, Single Data)
- Pipeline architectures: functional units perform different operations on the same data
- For fault tolerance, may want to execute same instructions redundantly to detect and mask errors – for task replication

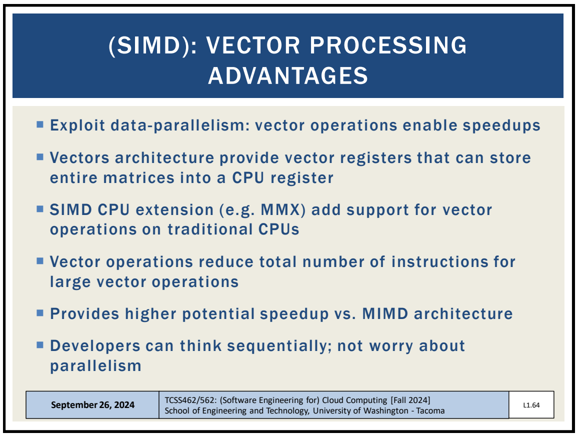
62



FLYNN'S TAXONOMY

- SISD (Single Instruction Single Data)
  Scalar architecture with one processor/core.
  - Individual cores of modern multicore processors are "SISD"
- SIMD (Single Instruction, Multiple Data)
  Supports vector processing
  - When SIMD instructions are issued, operations on individual vector components are carried out concurrently
  - Two 64-element vectors can be added in parallel
  - Vector processing instructions added to modern CPUs
  - Example: Intel MMX (multimedia) instructions

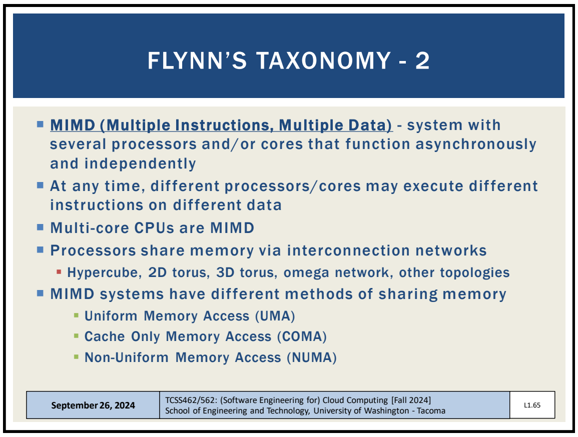
63



(SIMD): VECTOR PROCESSING ADVANTAGES

- Exploit data-parallelism: vector operations enable speedups
- Vectors architecture provide vector registers that can store entire matrices into a CPU register
- SIMD CPU extension (e.g. MMX) add support for vector operations on traditional CPUs
- Vector operations reduce total number of instructions for large vector operations
- Provides higher potential speedup vs. MIMD architecture
- Developers can think sequentially; not worry about parallelism

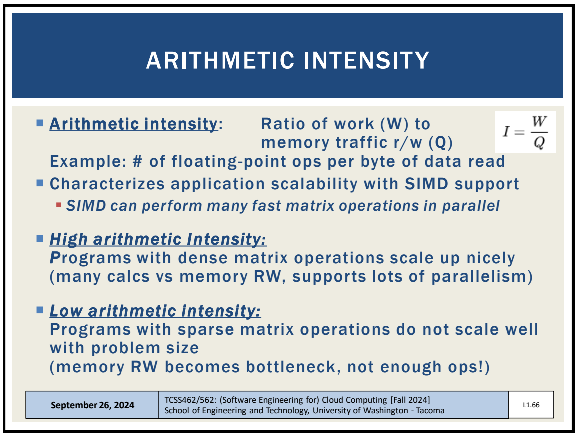
64



FLYNN'S TAXONOMY - 2

- MIMD (Multiple Instructions, Multiple Data) - system with several processors and/or cores that function asynchronously and independently
- At any time, different processors/cores may execute different instructions on different data
- Multi-core CPUs are MIMD
- Processors share memory via interconnection networks
  - Hypercube, 2D torus, 3D torus, omega network, other topologies
- MIMD systems have different methods of sharing memory
  - Uniform Memory Access (UMA)
  - Cache Only Memory Access (COMA)
  - Non-Uniform Memory Access (NUMA)

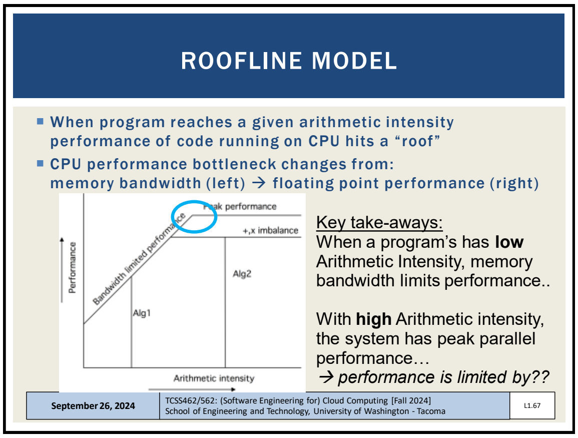
65



ARITHMETIC INTENSITY

- Arithmetic intensity: Ratio of work (W) to memory traffic r/w (Q) $I = \frac{W}{Q}$
  Example: # of floating-point ops per byte of data read
- Characterizes application scalability with SIMD support
  - *SIMD can perform many fast matrix operations in parallel*
- ***High arithmetic Intensity:***
  Programs with dense matrix operations scale up nicely (many calcs vs memory RW, supports lots of parallelism)
- ***Low arithmetic intensity:***
  Programs with sparse matrix operations do not scale well with problem size
  (memory RW becomes bottleneck, not enough ops!)

66



ROOFLINE MODEL

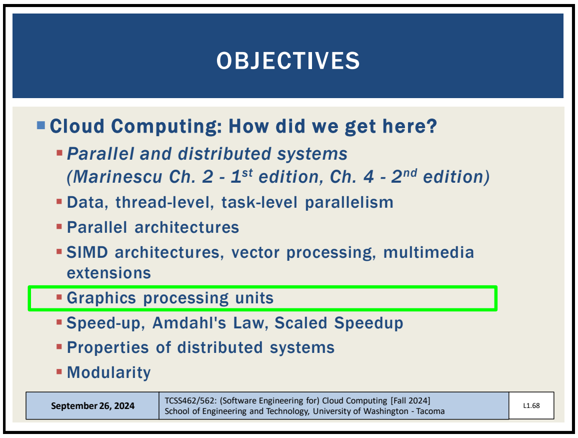
- When program reaches a given arithmetic intensity performance of code running on CPU hits a "roof"
- CPU performance bottleneck changes from: memory bandwidth (left) → floating point performance (right)

Peak performance
+,x imbalance
Bandwidth limited performance
Performance
Alg2
Alg1
Arithmetic intensity

Key take-aways:
When a program's has **low** Arithmetic Intensity, memory bandwidth limits performance..

With **high** Arithmetic intensity, the system has peak parallel performance…
→ *performance is limited by??*

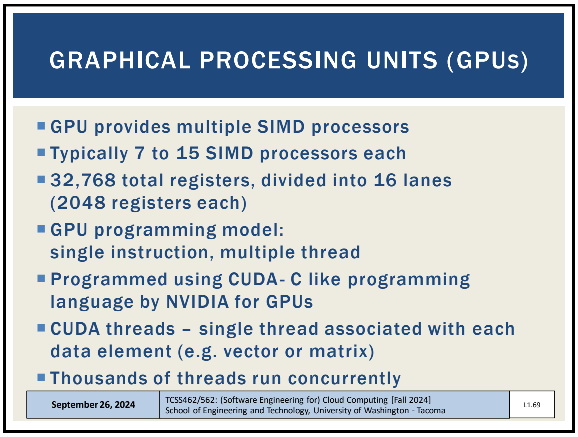
67



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

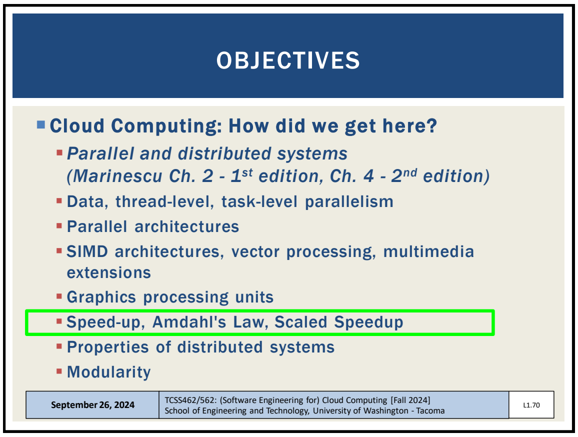
68



GRAPHICAL PROCESSING UNITS (GPUs)

- GPU provides multiple SIMD processors
- Typically 7 to 15 SIMD processors each
- 32,768 total registers, divided into 16 lanes (2048 registers each)
- GPU programming model: single instruction, multiple thread
- Programmed using CUDA- C like programming language by NVIDIA for GPUs
- CUDA threads – single thread associated with each data element (e.g. vector or matrix)
- Thousands of threads run concurrently

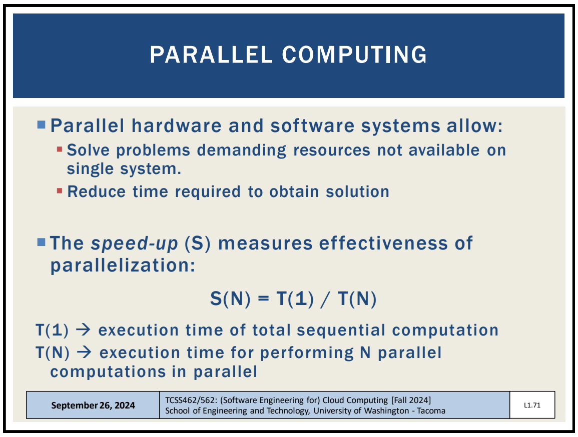
69



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

70



PARALLEL COMPUTING

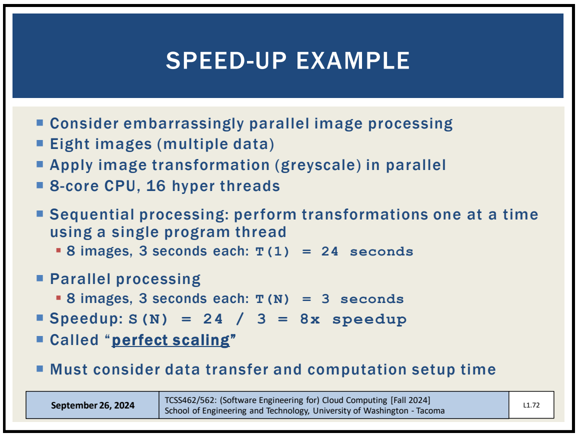
- Parallel hardware and software systems allow:
  - Solve problems demanding resources not available on single system.
  - Reduce time required to obtain solution

- The *speed-up* (S) measures effectiveness of parallelization:

$$S(N) = T(1) / T(N)$$

T(1) → execution time of total sequential computation
T(N) → execution time for performing N parallel computations in parallel

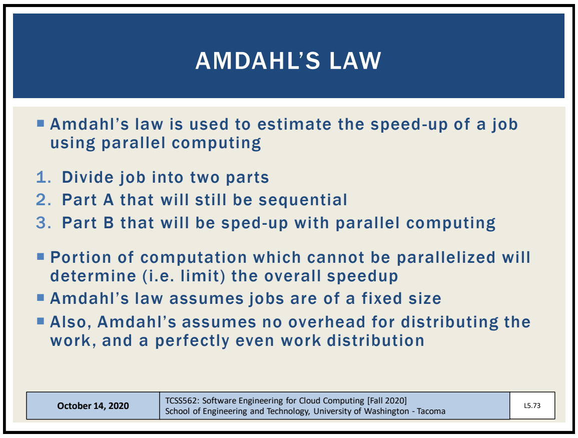
71



SPEED-UP EXAMPLE

- Consider embarrassingly parallel image processing
- Eight images (multiple data)
- Apply image transformation (greyscale) in parallel
- 8-core CPU, 16 hyper threads
- Sequential processing: perform transformations one at a time using a single program thread
  - 8 images, 3 seconds each: T(1) = 24 seconds
- Parallel processing
  - 8 images, 3 seconds each: T(N) = 3 seconds
- Speedup: S(N) = 24 / 3 = 8x speedup
- Called "**perfect scaling**"
- Must consider data transfer and computation setup time

72



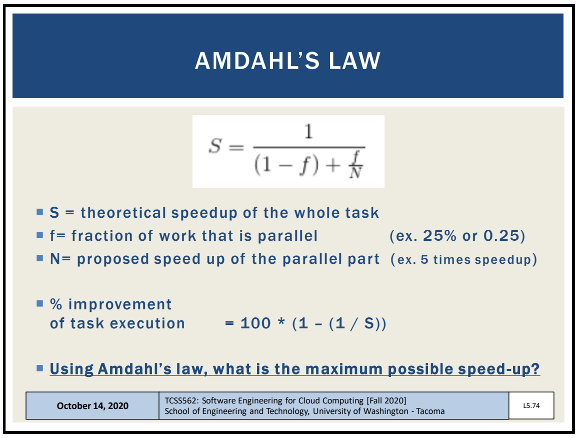
AMDAHL'S LAW

- Amdahl's law is used to estimate the speed-up of a job using parallel computing

1. Divide job into two parts
2. Part A that will still be sequential
3. Part B that will be sped-up with parallel computing

- Portion of computation which cannot be parallelized will determine (i.e. limit) the overall speedup
- Amdahl's law assumes jobs are of a fixed size
- Also, Amdahl's assumes no overhead for distributing the work, and a perfectly even work distribution

73



AMDAHL'S LAW

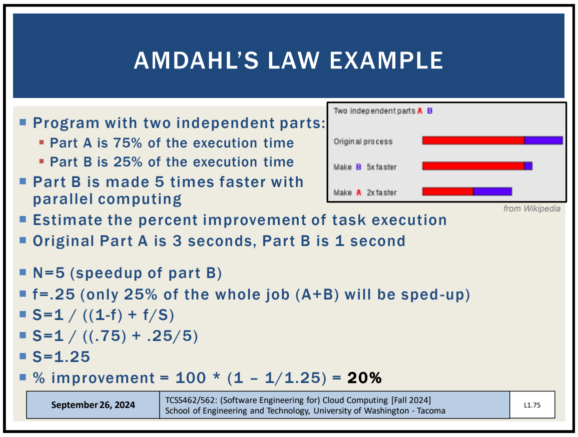
$$S = \frac{1}{(1-f) + \frac{f}{N}}$$

- S = theoretical speedup of the whole task
- f= fraction of work that is parallel (ex. 25% or 0.25)
- N= proposed speed up of the parallel part (ex. 5 times speedup)

- % improvement of task execution = 100 * (1 – (1 / S))

- **Using Amdahl's law, what is the maximum possible speed-up?**

74



AMDAHL'S LAW EXAMPLE

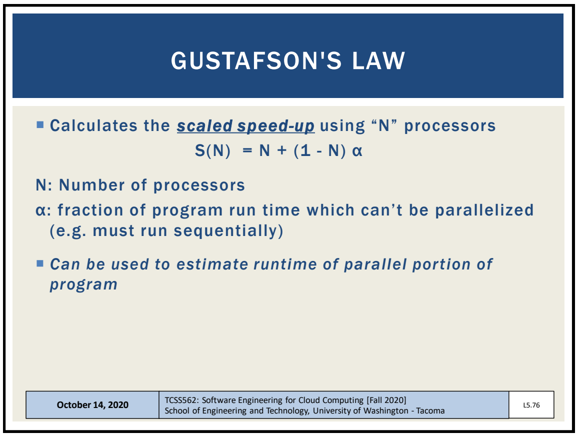
- Program with two independent parts:
  - Part A is 75% of the execution time
  - Part B is 25% of the execution time
- Part B is made 5 times faster with parallel computing
- Estimate the percent improvement of task execution
- Original Part A is 3 seconds, Part B is 1 second

Two independent parts A B
Original process
Make B 5x faster
Make A 2x faster

from Wikipedia

- N=5 (speedup of part B)
- f=.25 (only 25% of the whole job (A+B) will be sped-up)
- S=1 / ((1-f) + f/S)
- S=1 / ((.75) + .25/5)
- S=1.25
- % improvement = 100 * (1 – 1/1.25) = **20%**

75



GUSTAFSON'S LAW

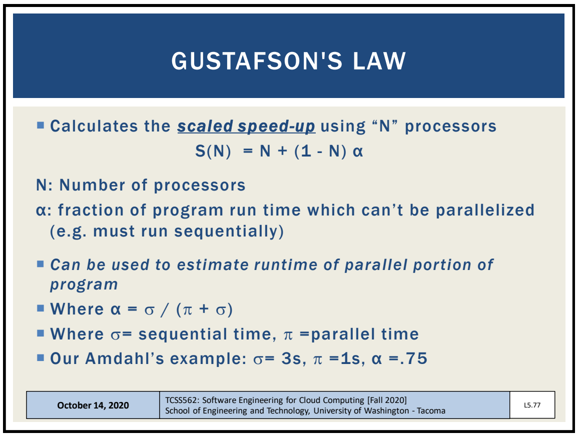
- Calculates the ***scaled speed-up*** using "N" processors
  S(N) = N + (1 - N) α

N: Number of processors

α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- *Can be used to estimate runtime of parallel portion of program*

76



GUSTAFSON'S LAW

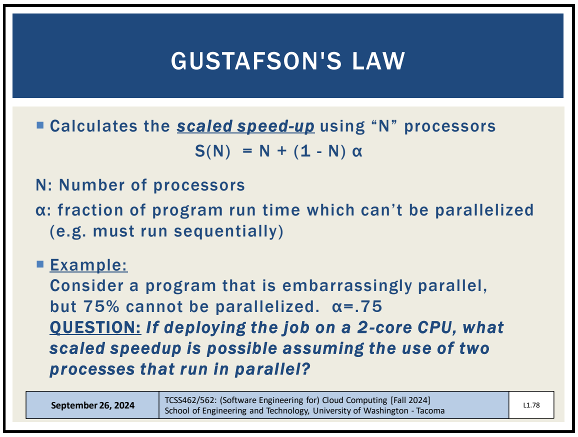
- Calculates the ***scaled speed-up*** using "N" processors
  S(N) = N + (1 - N) α

N: Number of processors

α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- *Can be used to estimate runtime of parallel portion of program*
- Where α = σ / (π + σ)
- Where σ= sequential time, π =parallel time
- Our Amdahl's example: σ= 3s, π =1s, α =.75

77



GUSTAFSON'S LAW

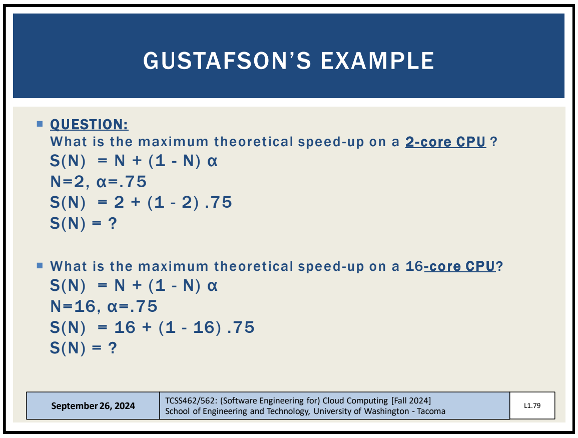
- Calculates the ***scaled speed-up*** using "N" processors
  S(N) = N + (1 - N) α

N: Number of processors

α: fraction of program run time which can't be parallelized (e.g. must run sequentially)

- Example:
  Consider a program that is embarrassingly parallel, but 75% cannot be parallelized. α=.75
  QUESTION: ***If deploying the job on a 2-core CPU, what scaled speedup is possible assuming the use of two processes that run in parallel?***

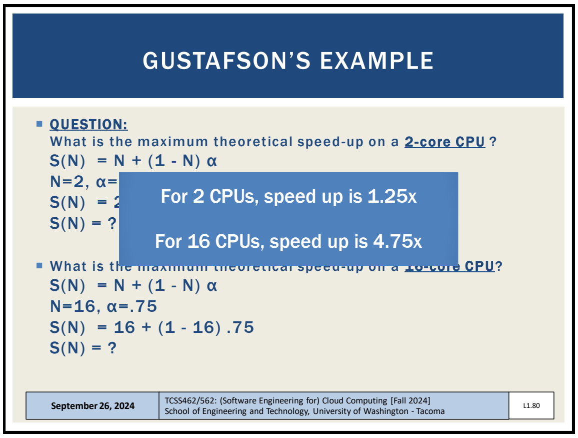
78



GUSTAFSON'S EXAMPLE

- **QUESTION:**
  What is the maximum theoretical speed-up on a **2-core CPU** ?
  $S(N) = N + (1 - N)\ \alpha$
  $N=2,\ \alpha=.75$
  $S(N) = 2 + (1 - 2)\ .75$
  $S(N) = ?$
- What is the maximum theoretical speed-up on a 16-**core CPU**?
  $S(N) = N + (1 - N)\ \alpha$
  $N=16,\ \alpha=.75$
  $S(N) = 16 + (1 - 16)\ .75$
  $S(N) = ?$

79



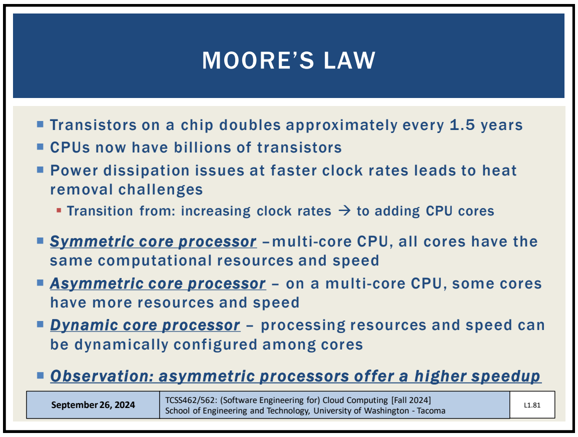
GUSTAFSON'S EXAMPLE

- **QUESTION:**
  What is the maximum theoretical speed-up on a **2-core CPU** ?
  $S(N) = N + (1 - N)\ \alpha$
  $N=2,\ \alpha=$
  $S(N) = 2$
  $S(N) = ?$
- What is the maximum theoretical speed-up on a **16-core CPU**?
  $S(N) = N + (1 - N)\ \alpha$
  $N=16,\ \alpha=.75$
  $S(N) = 16 + (1 - 16)\ .75$
  $S(N) = ?$

For 2 CPUs, speed up is 1.25x

For 16 CPUs, speed up is 4.75x

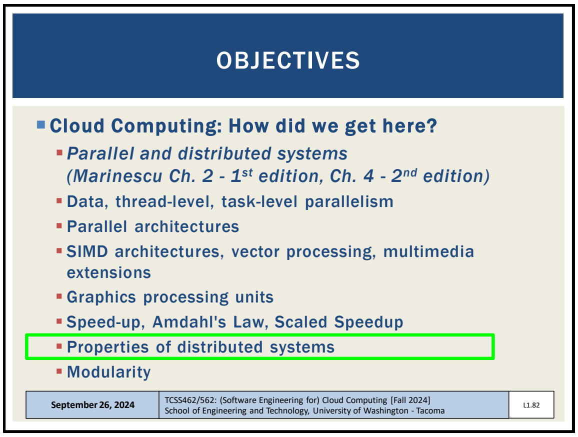
80



MOORE'S LAW

- Transistors on a chip doubles approximately every 1.5 years
- CPUs now have billions of transistors
- Power dissipation issues at faster clock rates leads to heat removal challenges
  - Transition from: increasing clock rates → to adding CPU cores
- ***Symmetric core processor*** –multi-core CPU, all cores have the same computational resources and speed
- ***Asymmetric core processor*** – on a multi-core CPU, some cores have more resources and speed
- ***Dynamic core processor*** – processing resources and speed can be dynamically configured among cores
- ***Observation: asymmetric processors offer a higher speedup***

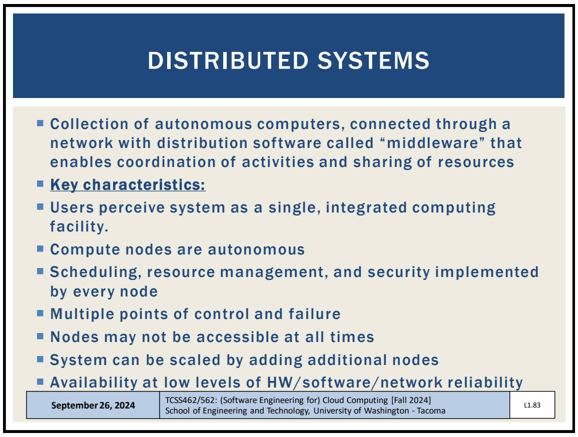
81



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

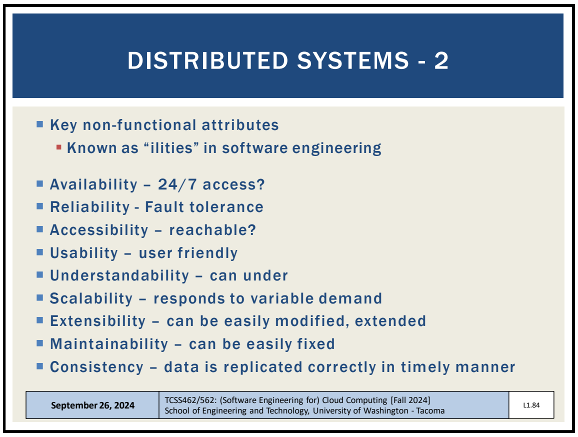
82



DISTRIBUTED SYSTEMS

- Collection of autonomous computers, connected through a network with distribution software called "middleware" that enables coordination of activities and sharing of resources
- **Key characteristics:**
- Users perceive system as a single, integrated computing facility.
- Compute nodes are autonomous
- Scheduling, resource management, and security implemented by every node
- Multiple points of control and failure
- Nodes may not be accessible at all times
- System can be scaled by adding additional nodes
- Availability at low levels of HW/software/network reliability

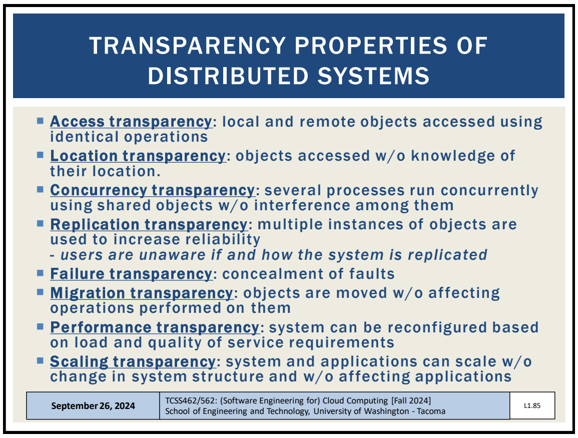
83



DISTRIBUTED SYSTEMS - 2

- Key non-functional attributes
  - Known as "ilities" in software engineering
- Availability – 24/7 access?
- Reliability - Fault tolerance
- Accessibility – reachable?
- Usability – user friendly
- Understandability – can under
- Scalability – responds to variable demand
- Extensibility – can be easily modified, extended
- Maintainability – can be easily fixed
- Consistency – data is replicated correctly in timely manner

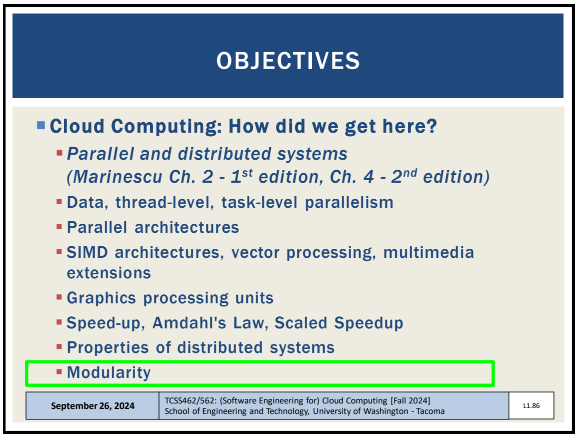
84



TRANSPARENCY PROPERTIES OF DISTRIBUTED SYSTEMS

- **Access transparency**: local and remote objects accessed using identical operations
- **Location transparency**: objects accessed w/o knowledge of their location.
- **Concurrency transparency**: several processes run concurrently using shared objects w/o interference among them
- **Replication transparency**: multiple instances of objects are used to increase reliability
  - users are unaware if and how the system is replicated
- **Failure transparency**: concealment of faults
- **Migration transparency**: objects are moved w/o affecting operations performed on them
- **Performance transparency**: system can be reconfigured based on load and quality of service requirements
- **Scaling transparency**: system and applications can scale w/o change in system structure and w/o affecting applications

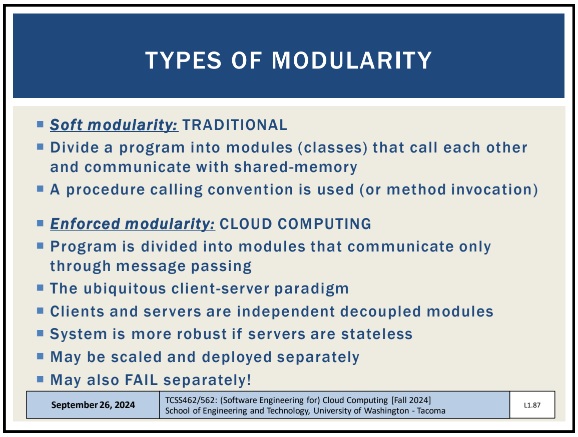
85



OBJECTIVES

- **Cloud Computing: How did we get here?**
  - *Parallel and distributed systems (Marinescu Ch. 2 - 1st edition, Ch. 4 - 2nd edition)*
  - Data, thread-level, task-level parallelism
  - Parallel architectures
  - SIMD architectures, vector processing, multimedia extensions
  - Graphics processing units
  - Speed-up, Amdahl's Law, Scaled Speedup
  - Properties of distributed systems
  - Modularity

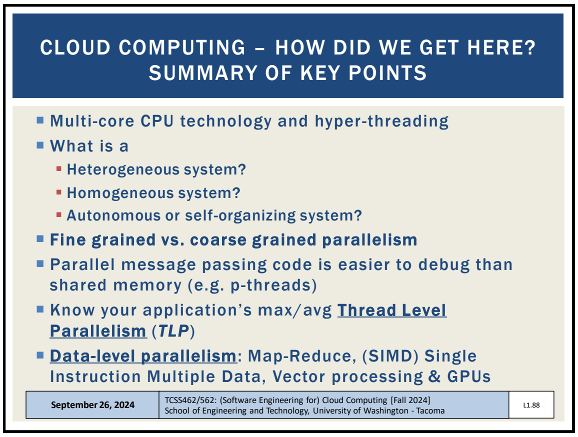
86



TYPES OF MODULARITY

- ***Soft modularity:*** TRADITIONAL
- Divide a program into modules (classes) that call each other and communicate with shared-memory
- A procedure calling convention is used (or method invocation)
- ***Enforced modularity:*** CLOUD COMPUTING
- Program is divided into modules that communicate only through message passing
- The ubiquitous client-server paradigm
- Clients and servers are independent decoupled modules
- System is more robust if servers are stateless
- May be scaled and deployed separately
- May also FAIL separately!

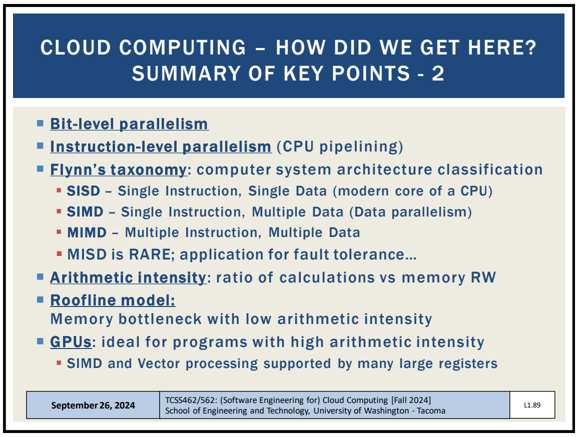
87



CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS

- Multi-core CPU technology and hyper-threading
- What is a
  - Heterogeneous system?
  - Homogeneous system?
  - Autonomous or self-organizing system?
- **Fine grained vs. coarse grained parallelism**
- Parallel message passing code is easier to debug than shared memory (e.g. p-threads)
- Know your application's max/avg **Thread Level Parallelism** (***TLP***)
- **Data-level parallelism**: Map-Reduce, (SIMD) Single Instruction Multiple Data, Vector processing & GPUs

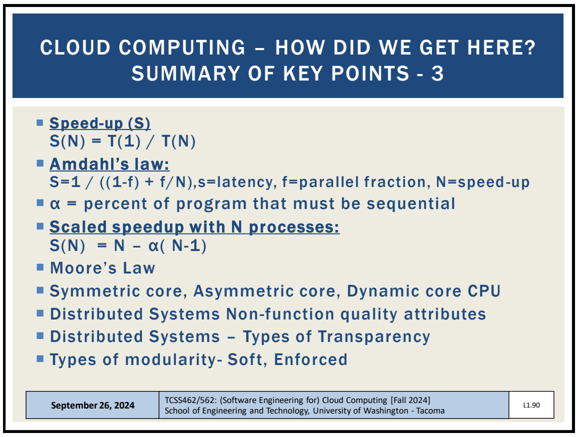
88



CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS - 2

- **Bit-level parallelism**
- **Instruction-level parallelism** (CPU pipelining)
- **Flynn's taxonomy**: computer system architecture classification
  - **SISD** – Single Instruction, Single Data (modern core of a CPU)
  - **SIMD** – Single Instruction, Multiple Data (Data parallelism)
  - **MIMD** – Multiple Instruction, Multiple Data
  - MISD is RARE; application for fault tolerance…
- **Arithmetic intensity**: ratio of calculations vs memory RW
- **Roofline model:**
  Memory bottleneck with low arithmetic intensity
- **GPUs**: ideal for programs with high arithmetic intensity
  - SIMD and Vector processing supported by many large registers

89



CLOUD COMPUTING – HOW DID WE GET HERE? SUMMARY OF KEY POINTS - 3

- **Speed-up (S)**
  S(N) = T(1) / T(N)
- **Amdahl's law:**
  S=1 / ((1-f) + f/N),s=latency, f=parallel fraction, N=speed-up
- α = percent of program that must be sequential
- **Scaled speedup with N processes:**
  S(N) = N – α(N-1)
- Moore's Law
- Symmetric core, Asymmetric core, Dynamic core CPU
- Distributed Systems Non-function quality attributes
- Distributed Systems – Types of Transparency
- Types of modularity- Soft, Enforced

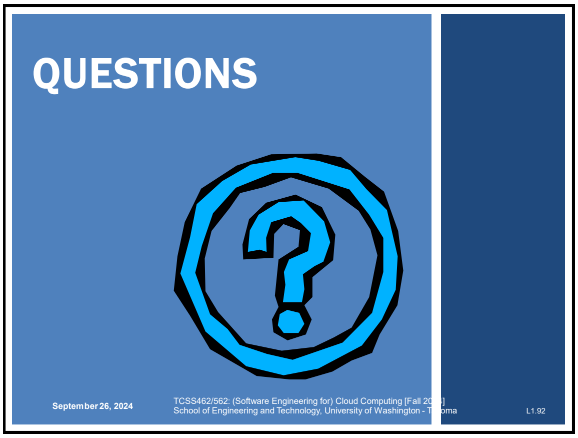
90

SEE YOU TUESDAY!

OCTOBER 1ST
3:40 PM
…

L1.91

91



QUESTIONS

September 26, 2024

TCSS462/562: (Software Engineering for) Cloud Computing [Fall 20]
School of Engineering and Technology, University of Washington - T oma

L1.92

92