UNIVERSITY OF COPENHAGEN

Advanced Computer Systems (ACS)

Dmitriy Traytel
Associate Professor, DIKU

Yongluan Zhou
Professor, DIKU

(Slides also by Marcos Vaz Salles)

Why study computer systems?

- The Microsoft/Oracle question

  How can I program large systems with clean interfaces and high performance?

- The Cloudera/Teradata question

  How do I build systems to process TBs to PBs of data?

- The Amazon/Google question

  How can I understand the guarantees and reliability of scalable services offered to me on the cloud?

What is the scale of our computer systems?

- Source: Michael Brodie, Computer Science 2.0, presented at VLDB 2007, Vienna, Austria
- **Databases**
  - AT&T has 11 exabytes (10^7 TB) of wireline, wireless, Internet data; 2+ trillion calls
  - Google's BigTable (US): 1-2 petabytes,
  - Wal-Mart (US): 500 TB, 10^7 transactions / day
  - All in 2004

What is the scale of our computer systems?

- Source: Michael Brodie, Computer Science 2.0, presented at VLDB 2007, Vienna, Austria

- **Internet**
  - 1/2 billion hosts (IP addresses)
  - 1.17 billion users (~5 billion today)
  - or 17.8% of the world's population
- **Web**
  - 109 million distinct web sites
  - 29.7 billion web pages
  - ~5 pages for every man, woman, and child on the planet
  - 7.2 billion Web searches/month (3.9 billion by Google) far exceed the world population

(> 1.9 billion websites, 150 billion Google searches/month today)

What is the scale of our computer systems?

- Source: Michael Brodie, Computer Science 2.0, presented at VLDB 2007, Vienna, Austria

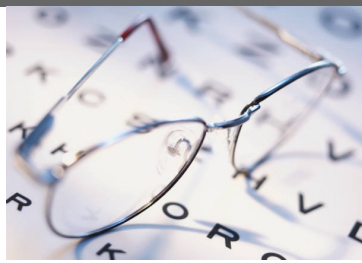
- **Facebook**
  - 1.8 billion photos
  - 41 million active users
  - 10^5 new users / day
  - 1,800 applications
  - (today > 2.8 billion, ~37% of world population)
- **YouTube**
  - 1.7 billion served / month
  - 1 million streams / day (today billions of views / day)

What is the scale of our computer systems?

- Source: Michael Brodie, Computer

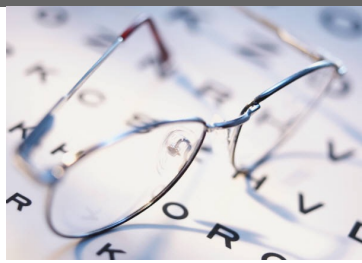
How can we think about and architect large-scale computer systems?

- 1.7 billion served / month
- 1 million streams / day
  (today billions of views / day)



What should we learn in this course?

- Knowledge
  - Describe the design of transactional and distributed systems, including techniques for modularity, performance, and fault tolerance
  - Explain how to employ strong modularity through a client-service abstraction as a paradigm to structure computer systems, while hiding complexity of implementation from clients
  - Explain techniques for large-scale data processing
- Skills
  - Implement systems that include mechanisms for modularity, atomicity, and fault tolerance
  - Structure and conduct experiments to evaluate a system's performance



What should we learn in this course?

- Competences
  - Discuss design alternatives for a modular computer system, identifying desired system properties as well as describing mechanisms for improving performance while arguing for their correctness
  - Analyze protocols for concurrency control and recovery, as well as for distribution and replication
  - Apply principles of large-scale data processing to analyze concrete information-processing problems

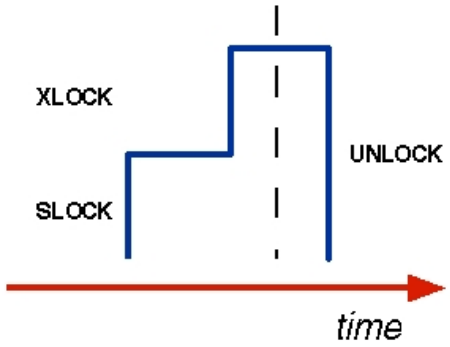
ACS: What will we study?

- Fundamentals
  - Abstractions: interpreters, memory, communication links
  - Modularity with clients and services, RPC
  - Techniques for performance, e.g., concurrency, fast paths, dallying, batching, speculation

Property: Strong Modularity

ACS: What will we study?

- Concurrency Control and Recovery
  - Two-phase locking
  - Serializability, schedules
  - Optimistic and multi-version approaches to concurrency control
  - Recovery concepts
  - ARIES recovery algorithm



Properties:

Atomicity, Isolation, and Durability

ACS: What will we study?

- Experimental Design
  - Performance metrics, workloads
  - Structuring and conducting simple experiments

ACS: What will we study?

- Reliability & Distribution
  - Reliability concepts
  - Replication techniques
  - Topics in coordination and distributed transactions
- Communication
  - Message queues, BASE
  - End-to-end argument

  (if time allows)

Property: High Availability

ACS: What will we study?

- Data Processing
  - Operators
  - External sorting
  - Hash- and sort-based techniques for multiple operations (e.g., set operations, joins)
  - Parallelism

Property: Scalability with Data Size

References & Course Materials

- Course webpage
  - Kurser: http://kurser.ku.dk/course/ndak15006u/
- Course materials in Absalon (Canvas)
  - Tentative course syllabus
    - Includes readings for after each class
  - Slides before each class
  - Last year's video recordings
  - Assignments & Feedback
  - Discussion forums
- Please always post your questions in Absalon
  - Your colleagues can profit too!

References & Course Materials

- Book
  - Advanced Computer Systems (ACS): DIKU Course Compendium. Collected references from sources cited therein, organized by Marcos Vaz Salles and Michael Kirkedal Carøe.
  - Same compendium as previous year!
- Papers & other references
  - Several listed in the syllabus
  - A few more will come as we go
  - Optional references and videos from recent conferences in computer systems for more depth

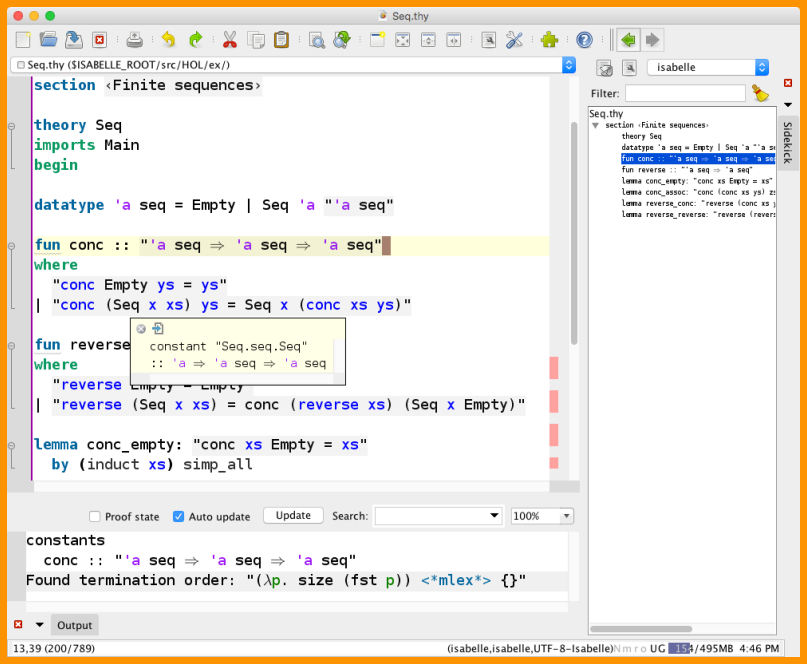
Team

- Lecturers
  - Dmitriy Traytel traytel@di.ku.dk
  - Yongluan Zhou zhou@di.ku.dk
  - Office hours:
    - By email appointment
- TAs
  - Benjamin Paddags
  - Cathy Mitchelmore
  - Christian Påbøl Jacobsen
  - Lennard Reese
  - Philipp Theyssen
  - Leonardo Lima
  - Tilman Zuckmantel
  - Meet them in the TA sessions on Thursdays!

Lecturer: Dmitriy Traytel

Le ytel

Interactive Theorem Proving



1988

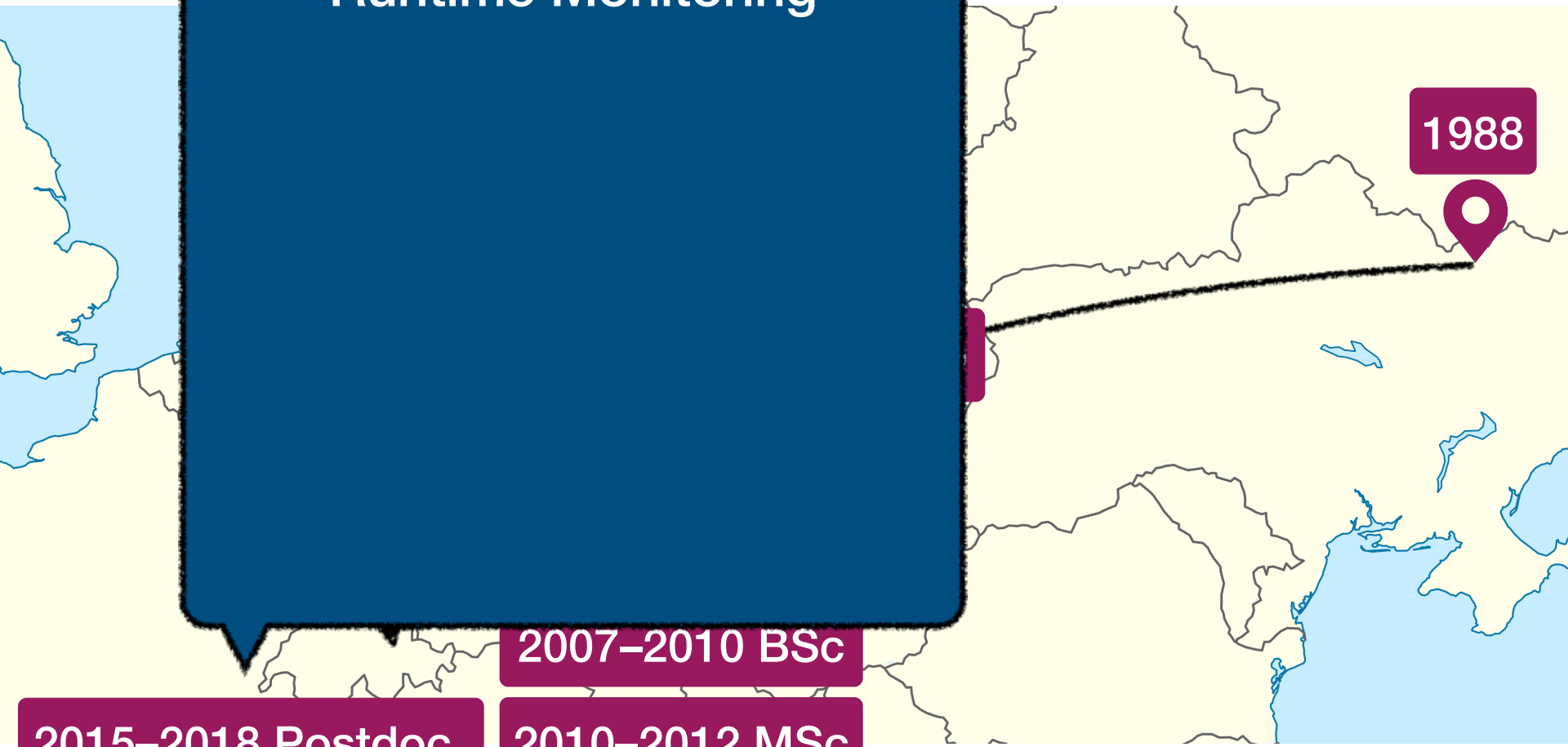
2007–2010 BSc

2010–2012 MSc

2012–2015 PhD

Technische Universität München

Lecturer: Dmitriy Traytel



ETH zürich

Technische Universität München

Lecturer: Dmitriy Traytel

Runtime Monitoring

time

1988

2007–2010 BSc

2010–2012 MSc

2012–2015 PhD

2015–2018 Postdoc

2018–2020 Oberass.

ETH zürich

Technische Universität München

Lecturer: Dmitriy Traytel

Runtime Monitoring

time

$$\Box \forall c.\ \forall s.\ ssh\_login(c, s) \wedge ((\Diamond_{[1min,20min]}\ net(c)) \wedge \Box_{[0,1d]}\ (\blacksquare_{=0}\ net(c) \rightarrow \Diamond_{[1min,20min]}\ net(c))) \rightarrow \Diamond_{[0,1d)}\ \blacklozenge_{=0}\ ssh\_logout(c, s)$$

property

1988

2007–2010 BSc

2010–2012 MSc

2012–2015 PhD

2015–2018 Postdoc

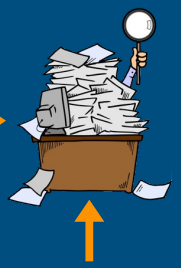
2018–2020 Oberass.

ETH zürich

Technische Universität München

Lecturer: Dmitriy Traytel

Runtime Monitoring



time

$$\Box\forall c.\ \forall s.\ ssh\_login(c, s) \wedge ((\Diamond_{[1min,20min]}\ net(c)) \wedge \Box_{[0,1d]}\ (\blacksquare_{=0}\ net(c) \rightarrow \Diamond_{[1min,20min]}\ net(c))) \rightarrow \Diamond_{[0,1d)}\ \blacklozenge_{=0}\ ssh\_logout(c, s)$$

property

1988

2007–2010 BSc

2010–2012 MSc

2012–2015 PhD

2015–2018 Postdoc

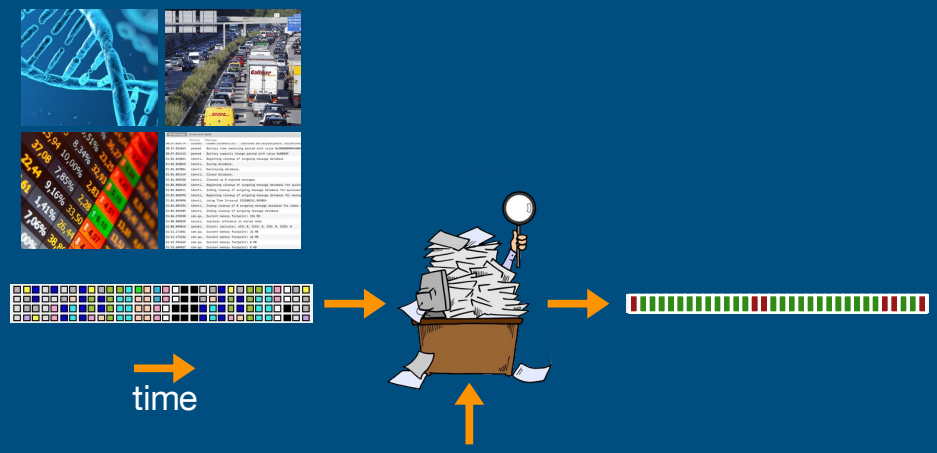
2018–2020 Oberass.

ETH zürich

Technische Universität München

Lecturer: Dmitriy Traytel

Runtime Monitoring



$$\Box \forall c.\ \forall s.\ ssh\_login(c, s) \wedge ((\Diamond_{[1min,20min]}\ net(c)) \wedge \Box_{[0,1d]}\ (\blacksquare_{=0}\ net(c) \rightarrow \Diamond_{[1min,20min]}\ net(c))) \rightarrow \Diamond_{[0,1d)}\ \blacklozenge_{=0}\ ssh\_logout(c, s)$$

1988

2007–2010 BSc

2010–2012 MSc

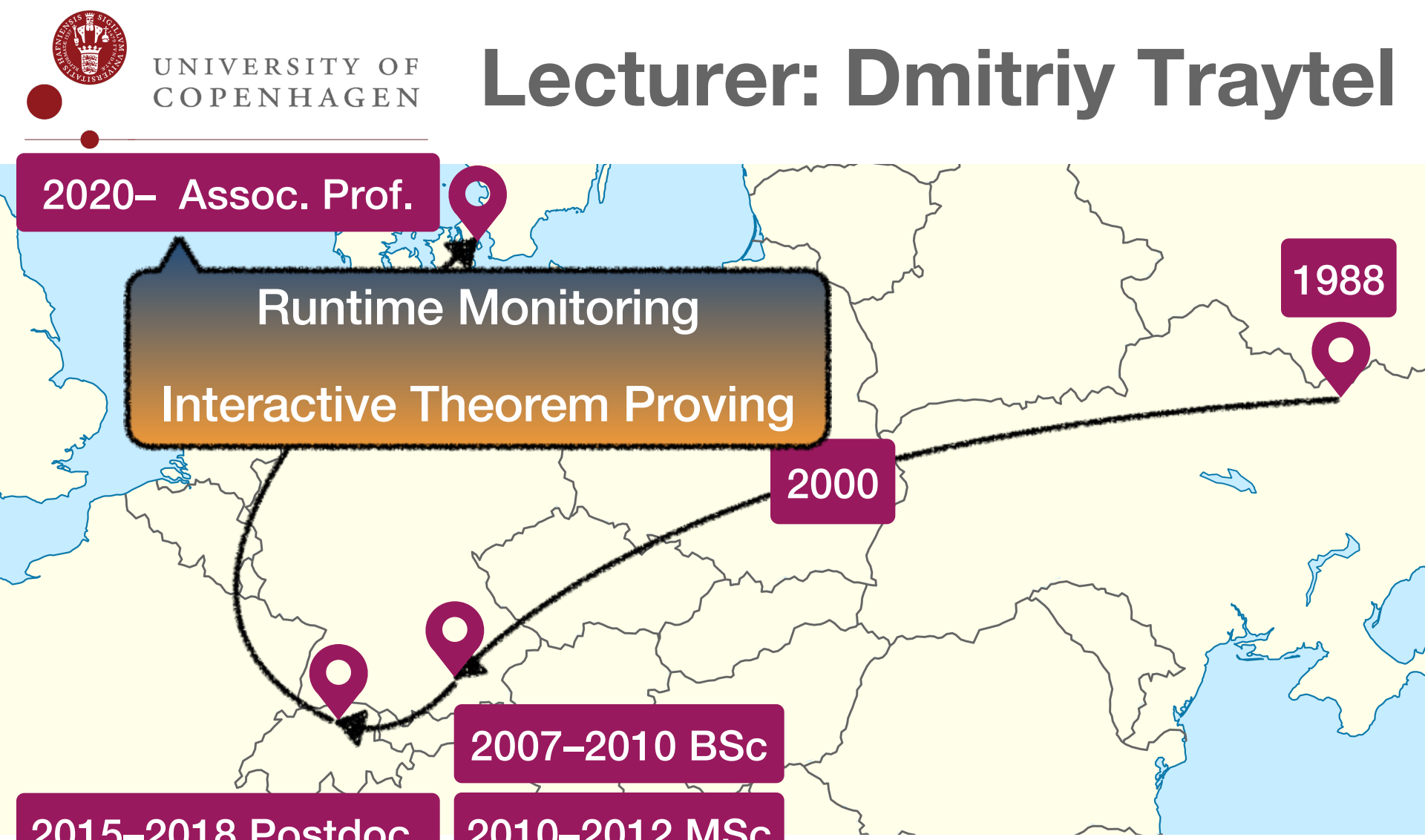
2012–2015 PhD

2015–2018 Postdoc

2018–2020 Oberass.

ETH zürich

Technische Universität München


UNIVERSITY OF COPENHAGEN
Lecturer: Dmitriy Traytel
2020– Assoc. Prof.
Runtime Monitoring
Interactive Theorem Proving
1988
2000
2007–2010 BSc
2015–2018 Postdoc
2010–2012 MSc
2018–2020 Oberass.
2012–2015 PhD
ETH zürich
Technische Universität München

Lecturer: Yongluan Zhou

- Professor, University of Copenhagen (**DIKU**)
  - Postdoc: EPFL (École Polytechnique Fédérale de Lausanne), Switzerland
  - PhD: National University of Singapore
- **Expertise:** Data Management
  - Big Data Systems
  - Database systems
- Co-leader of **Data Management Systems (DMS) Group**
- **Selected Ongoing Projects**
  - CEEDA: Consistent and Efficient Event-Driven Architecture
  - PAPRiCaS: Programming technology foundations for Accountability, Privacy-by-design & Robustness in Context-aware Systems

Weekly Schedule

- Lectures Tuesdays and Thursdays, 10am-12pm
  - Two 45 min sessions, 15 min break, with lecturer
  - Participation is encouraged ☺
- TA sessions (5 in-person)
- 4 x Thu 1pm-4pm + 1 x Thu 3pm-6pm
  - TAs will guide most of those
    - Exercises
    - **Feedback** on assignments
    - Assignment work time and Q&A
  - A lecturer will come over for some time, walk between classes to answer questions

Weekly Schedule

- Lectures Tuesdays and Thursdays, 10am-12pm
  - Two 45 min sessions, 15 min break, with lecturer
  - Participation is encouraged ☺

- TA sessions (5 in-person)
- 4 x Thu 1pm-4pm + 1 x Thu 3pm-6pm
  - TAs will guide most of those
    - Exercises
    - **Feedback** on assignments
    - Assignment work time and Q&A
  - A lecturer will come over for some time, walk between classes to answer questions

Learning is the main goal!

First Steps

- **Java Warm-up Exercise**
  - If you passed Advanced Java, you do not need it ☺
  - If you did not take Advanced Java, the assignments can be used as warm-up to tell you the level of Java you **need** for this course
- **Group formation**
  - Two kinds of groups
    - **TA session groups (~ 30 students)**
    - **Assignment groups (1-3 students)**
  - Groups of 3 students recommended for assignments
  - Register for both group kinds on Absalon
    - **Attend the TA session you are registered for**
    - Makes sense to find partners from the same TA session
    - TAs will help people who still need to find groups

Assignments

- **6 take-home assignments**
  - **3 Programming Assignments + 3 Theory Assignments**
  - **Groups: 3 people** strongly recommended
  - Programming Assignment 2 worth two points; all others one point each
  - **5/7 points** must be passed **to qualify** for exam
  - **Due dates**
    - **Programming Assignments: Dec 1, Dec 15, Jan 5**
    - **Theory Assignments: Dec 8, Dec 22, Jan 12**
  - **Resubmission** of Programming and Theory Assignments 1 and 2 possible (see syllabus for deadlines), but
    - You should give the assignments **an honest attempt** to be allowed to resubmit, i.e., you should submit enough that you can get feedback on to address in resubmission

Exam

- **Exam format**
  - 4-hour written exam (ITX) under invigilation with external grading on 7-point scale
  - Exam date is **January 26**
  - The exam is open-book, all written aids allowed
  - **The exam must be solved individually**

Exam

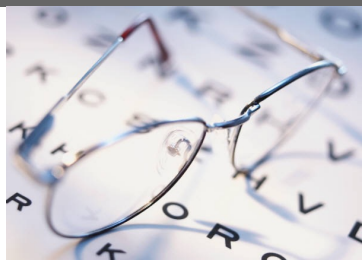
- **Exam format**
  - 4-hour written exam (ITX) under invigilation with external grading on 7-point scale
  - Exam date is **January 26**
  - The exam is open-book, all written aids allowed
  - **The exam must be solved individually**

Academic Integrity
taken very seriously

Acknowledgements

- Many of the slides in this course are based on or reproduce material kindly made available by Jerome Saltzer & M. Frans Kaashoek & Robert Morris (MIT, PCSD textbook material), Johannes Gehrke (Cornell/Microsoft, Ramakrishnan & Gehrke textbook), Gustavo Alonso (ETH Zurich, EAI course), Michael Freedman (Princeton), Nesime Tatbul (Intel Labs/MIT), James Kurose & Keith Ross (U Mass Amherst & NYU, networking textbook), Jens Dittrich (Saarland University)

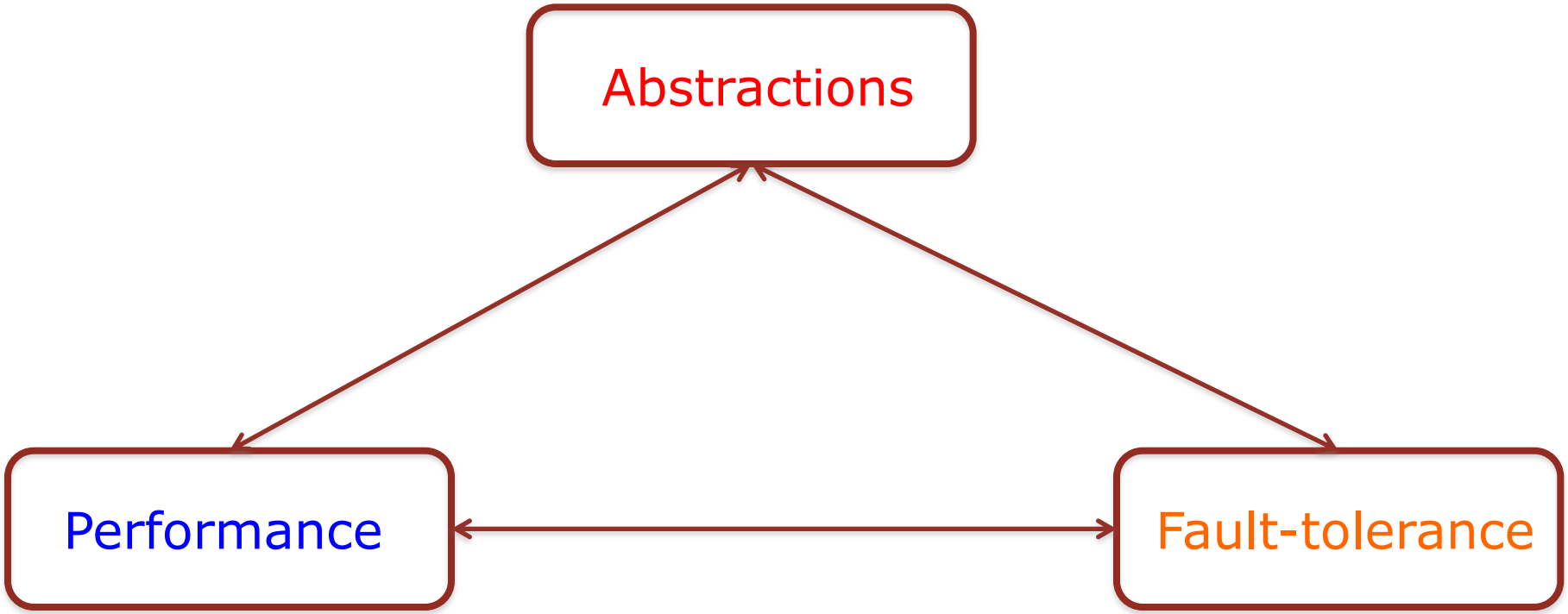
Questions so far?



What should we learn today?

- Identify the fundamental abstractions in computer systems and their APIs, including memory, interpreters, communication links
- Explain how names are used in the fundamental abstractions
- Design a top-level abstraction, respecting its correspondent API, based on lower-level abstractions
- Discuss performance and fault-tolerance aspects of such a design

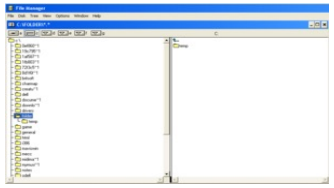
The Central Trade-off: Abstractions, Performance, Fault-Tolerance



Fundamental abstractions

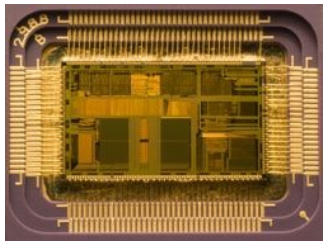
- ***Memory***
  - Read/write





- ***Interpreters***
  - Instruction repertoire
  - Environment
  - Instruction pointer

(loop (print (eval (read))))

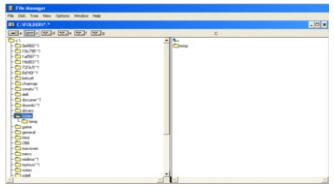


- ***Communication links***
  - Send/receive

Source: Saltzer & Kaashoek & Morris (partial)

Fundamental abstractions

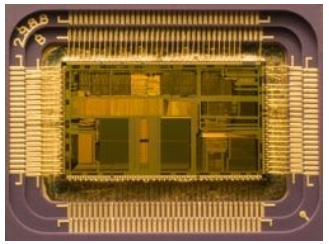
- ***Memory***
  - Read/write



- ***Interpreters***
  - Instruction repertoire
  - Environment
  - Instruction pointer

(loop (print (eval (read))))

Names make connections



- ***Communication links***
  - Send/receive

Source: Saltzer & Kaashoek & Morris (partial)

Examples of Names

- R1
- 1742
- 18.7.22.69
- web.mit.edu
- http://web.mit.edu/6.033
- 6.033-staff@mit.edu
- amsterdam
- /mit/6.033/www
- foo.c
- .. (as in cd .. or ls ..)
- wc
- (617)253-7149, x37149
- 021-84-2030

Source: Saltzer & Kaashoek & Morris (partial)

Examples of Names

- R1
- 1742
- 18.7.22.69
- web.mit.edu
- http://web.mit.edu/6.033
- 6.033-staff@mit.edu
- amsterdam
- /mit/6.033/www
- foo.c
- .. (as in cd .. or ls ..)
- wc
- (617)253-7149, x37149
- 021-84-2030

address is overloaded
name with location info
(e.g., LOAD 1742, R1)

Source: Saltzer & Kaashoek & Morris (partial)

Examples of Names

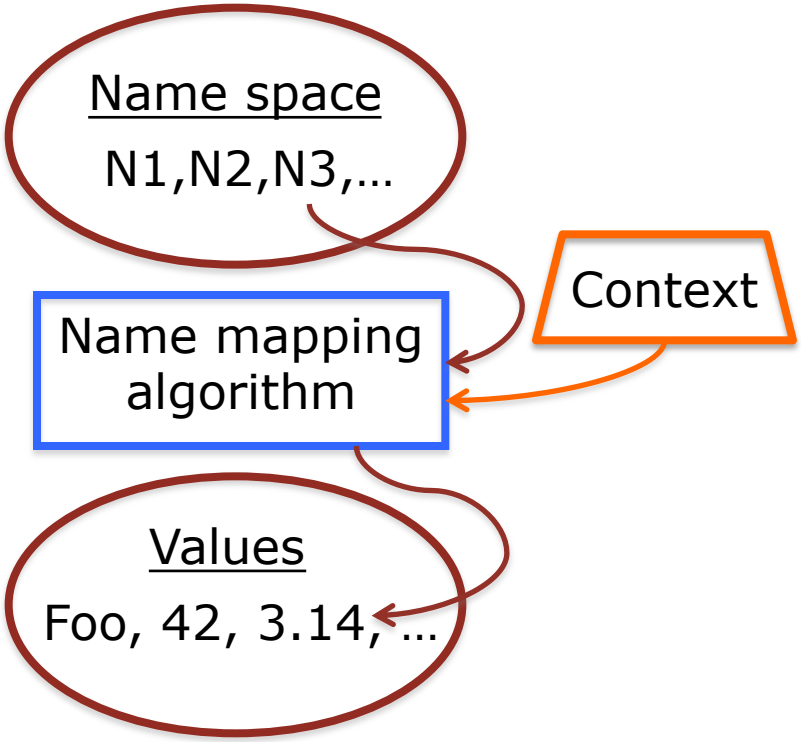
- R1
- 1742
- 18.7.22.69
- web.mit.edu
- http://web.mit.edu/6.033
- 6.033-staff@mit.edu
- amsterdam
- /mit/6.033/www
- foo.c
- .. (as in cd .. or ls ..)
- wc
- (617)253-7149, x37149
- 021-84-2030

address is overloaded name with location info (e.g., LOAD 1742, R1)

Names require a mapping scheme

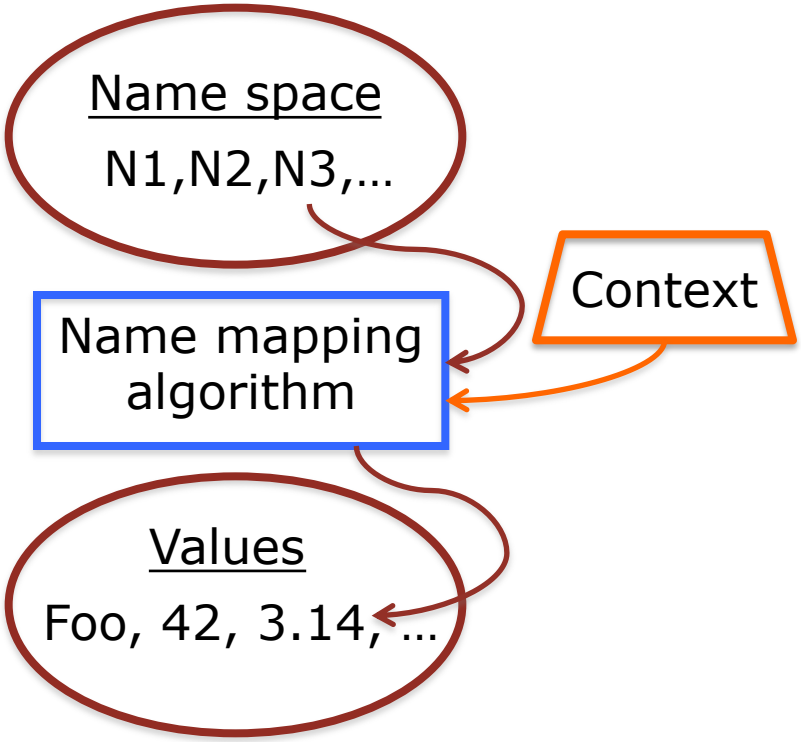
Source: Saltzer & Kaashoek & Morris (partial)

Name Mapping



- How can we map names?

Name Mapping

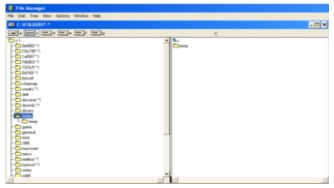


- How can we map names?
- Table lookup
  - Files inside directories
- Recursive lookup
  - Path names in file systems or URLs
- Multiple lookup
  - Java class loading

Fundamental abstractions

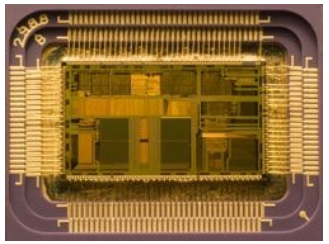
- ***Memory***
  - Read/write





- ***Interpreters***
  - Instruction repertoire
  - Environment
  - Instruction pointer

(loop (print (eval (read))))



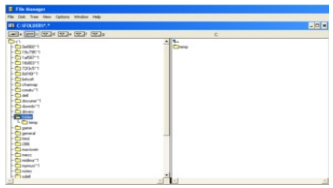
- ***Communication links***
  - Send/receive

Source: Saltzer & Kaashoek & Morris (partial)

Memory

- ***Memory***
  - `READ(name)` → `value`
  - `WRITE(name, value)`





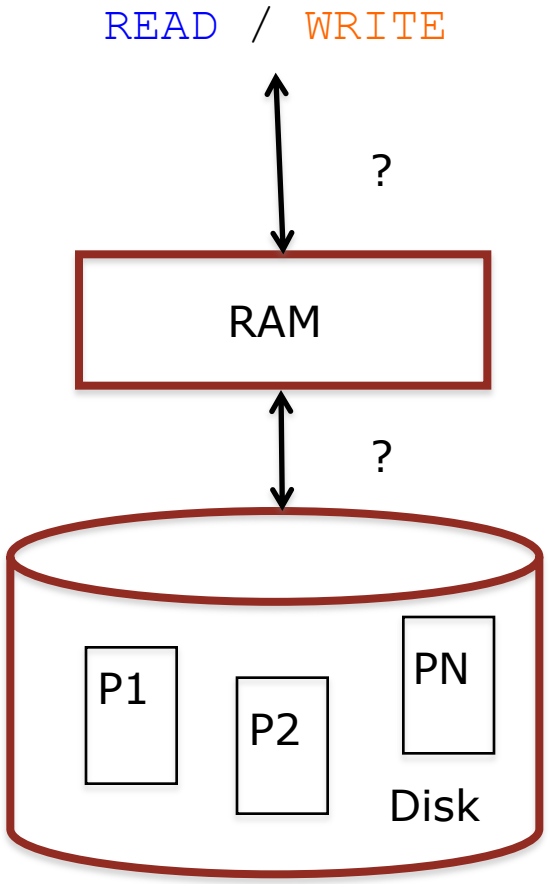
- Examples of Memory
  - Physical memory (RAM)
  - Multi-level memory hierarchy (registers, caches, RAM, flash, disk, tape)
  - Address spaces and virtual memory with paging
  - Key-value stores (e.g., Cassandra, Dynamo)
  - Database storage engines

Source: Saltzer & Kaashoek & Morris (partial)

How would you design a two-level memory abstraction consolidating disk and RAM?

- Characteristics of storage technologies
  - **RAM**: high cost per gigabyte, low latency, volatile
  - **Disk**: low cost per gigabyte, high latency, nonvolatile

- Design top-level abstraction respecting *Memory API*
- Abstraction must:
  - Address as much data as fits in disk
  - Use fixed-size pages for disk transfers
  - Use RAM efficiently to provide for low latency (on average)
  - Neither disk nor memory directly exposed, only READ/WRITE API



Write the pseudocode down!

Address Space Mapping

- Address spaces modular way to multiplex memory
- Naming scheme translating virtual into physical addresses
- Page map
  - Updated by kernel code
  - Lookup implemented in hardware
  - Concerns: Protection (Pr), representation, efficiency

Page Map

Page#	Block#	Pr
42	2	RX
53	45	R
64	97	RW
…	…	…

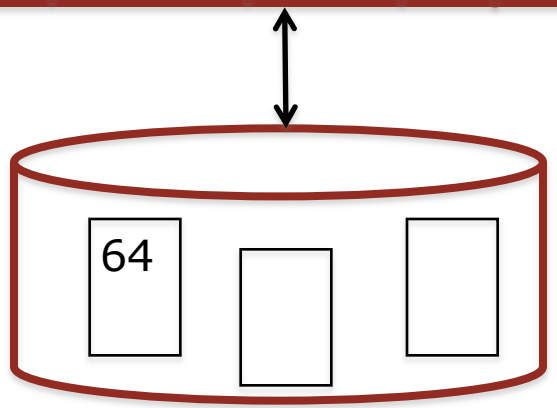
Address Space Mapping: Introducing Disks

- Use disk to store more blocks
- Pages may be either in memory or on disk

- Resident bit (R)
  - Access to non-resident pages results in page faults
- Page Fault
  - An indirection exception for missing pages

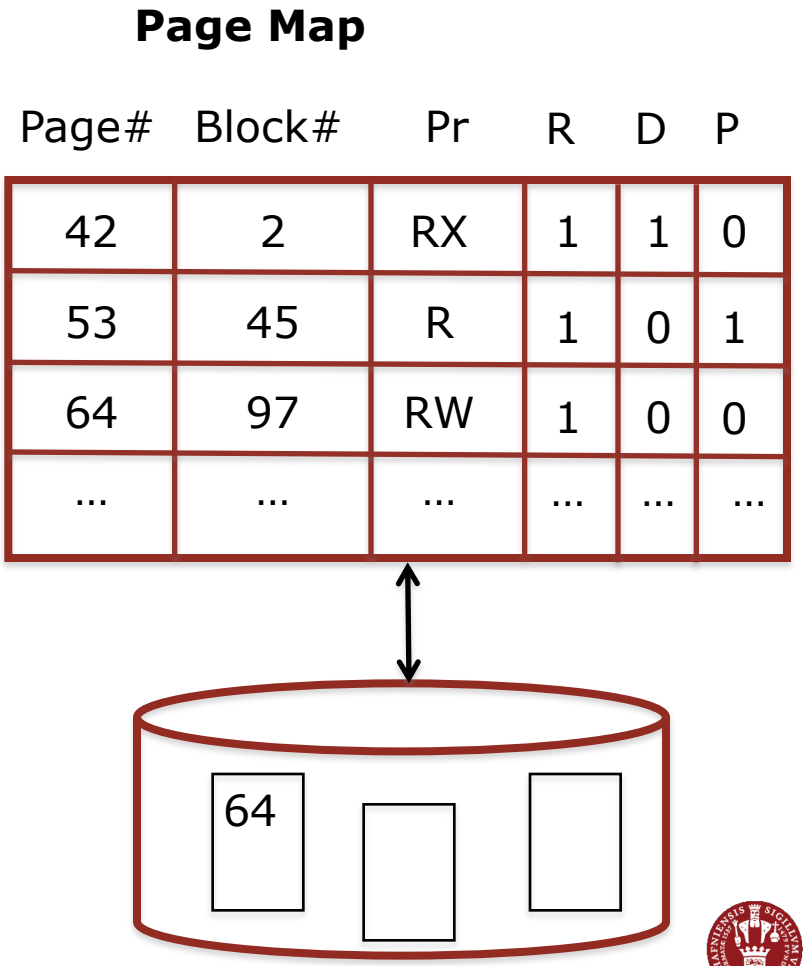
Page Map

Page#	Block#	Pr	R	D	P
42	2	RX	1	1	0
53	45	R	1	0	1
64	(D-42)	RW	0	0	0
…	…	…	…	…	…



Address Space Mapping: Introducing Disks

- Handling page faults
  - Trap to OS handler
  - Handler loads block from disk and updates mapping
  - If memory full, must choose some *victim* block for replacement
  - Page replacement algorithm, e.g., LRU
- Other metadata
  - *Dirty bit (D)*: Only write page back when it has changed!
  - *Pin bit (P)*: do not remove certain pages (e.g., code of OS handler itself)

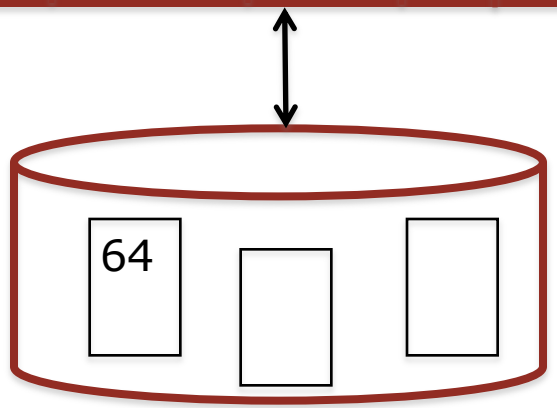


Virtual Memory with Paging: Abstractions, Performance, Fault-Tolerance

- ***Abstraction:*** Do we have any guarantees on two concurrent threads writing to the same memory?
- ***Performance:*** Do we get average latency close to RAM latency?
- ***Fault-Tolerance:*** What happens on failure? Do we have any guarantees about the state that is on disk?

Page Map

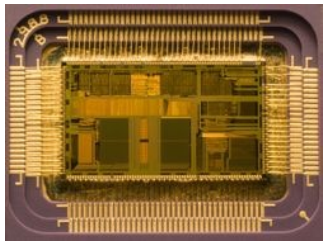
Page#	Block#	Pr	R	D	P
42	2	RX	1	1	0
53	45	R	1	0	1
64	97	RW	1	0	0
…	…	…	…	…	…



Interpreters

- ***Interpreter***
  - Instruction repertoire
  - Environment
  - Instruction pointer

(loop (print (eval (read))))



```
procedure INTERPRET()
        do forever
                instruction ← READ(instruction_pointer)
                perform instruction in environment context
                if interrupt_signal = TRUE then
                instruction_pointer ← entry of INTERRUPT_HANDLER
                environment ← environment of INTERRUPT_HANDLER
```

- Examples of Interpreters
  - Processors (CPU)
  - Programming language interpreters
  - Frameworks, e.g., MapReduce or Spark
  - Your own (layered) programs! (RPCs)

Source: Saltzer & Kaashoek & Morris (partial)

Communication links

- ***Communication links***
  - `SEND(linkName, outgoingMessageBuffer)`
  - `RECEIVE(linkName, incomingMessageBuffer)`

- Examples of Communication Links
  - Ethernet interface
  - IP datagram service
  - TCP sockets
  - Message-Oriented Middleware (MOM)
  - Multicast (e.g., CATOCS: Causal and Totally-Ordered Communication System)

Source: Saltzer & Kaashoek & Morris (partial)

Memory, Interpreters, Communication Links: Is that all there is?

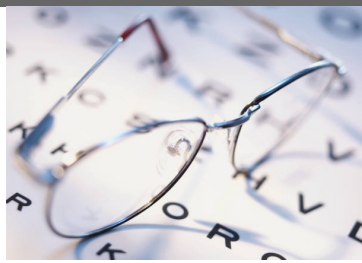
- Other abstractions also useful!

- ***Synchronization***
  - Locks
  - Condition variables & monitors

  (see, e.g., Chubby lock service from Google)

- ***Data processing***
  - Data transformations
  - Operators

  (see, e.g., data-parallel implementations)



What should we learn today?

- Identify the fundamental abstractions in computer systems and their APIs, including memory, interpreters, communication links
- Explain how names are used in the fundamental abstractions
- Design a top-level abstraction, respecting its correspondent API, based on lower-level abstractions
- Discuss performance and fault-tolerance aspects of such a design