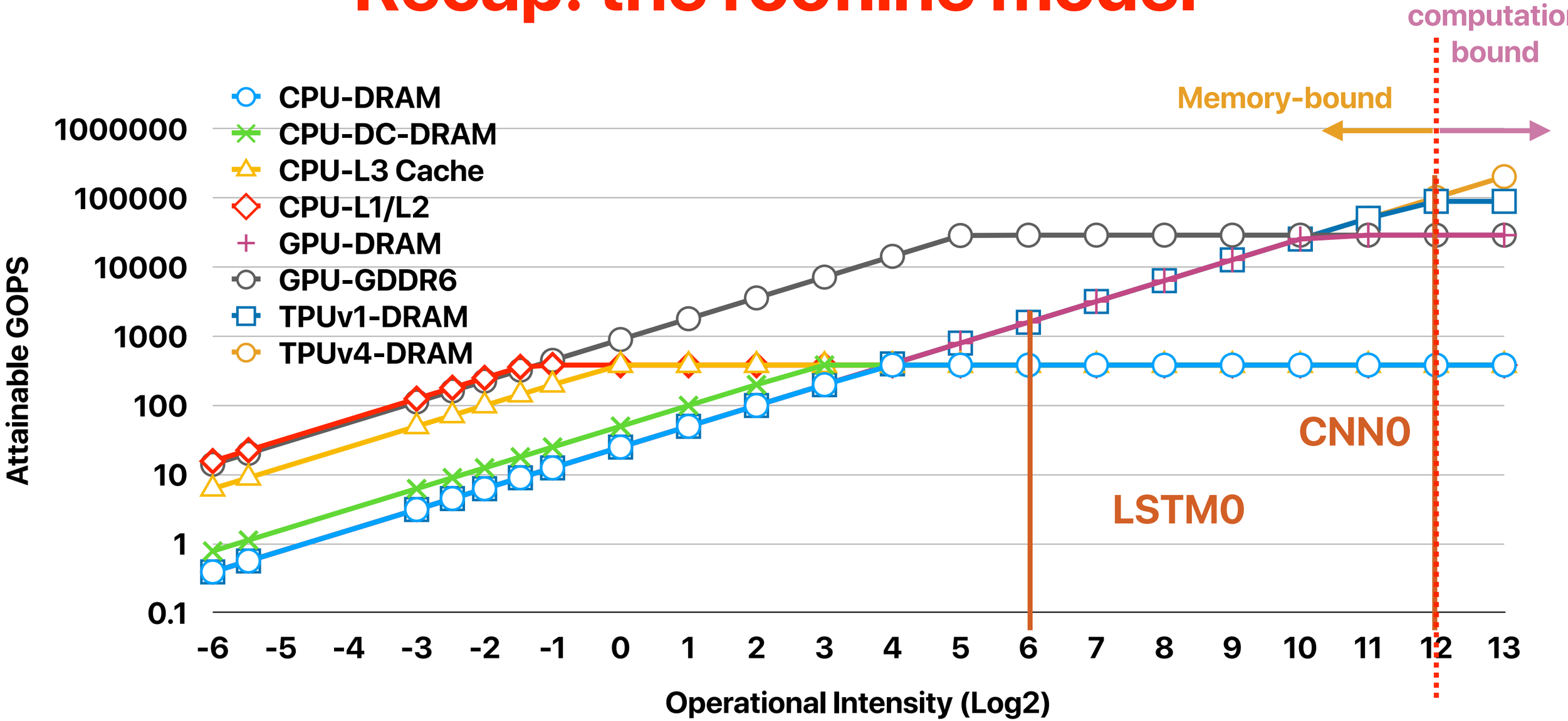
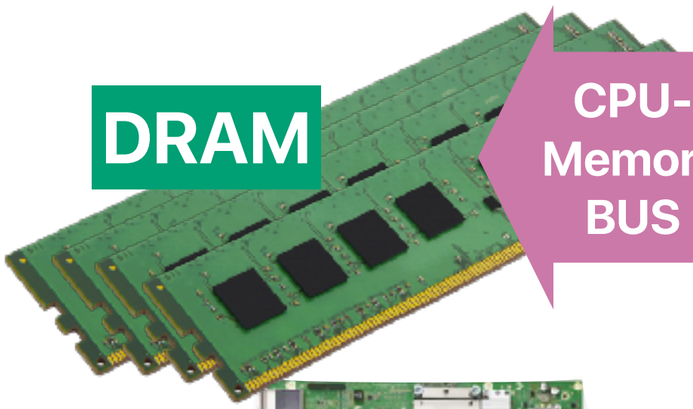
Retrospective: end-to-end arguments and hints in computer system designs in data-centric computing

Hung-Wei Tseng

Recap: the roofline model



Recap: The landscape of modern computers



CPU-Memory BUS

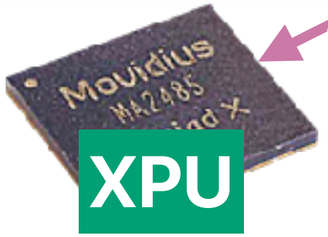
DDR4

Memory Controller

High-performance, but very specialized

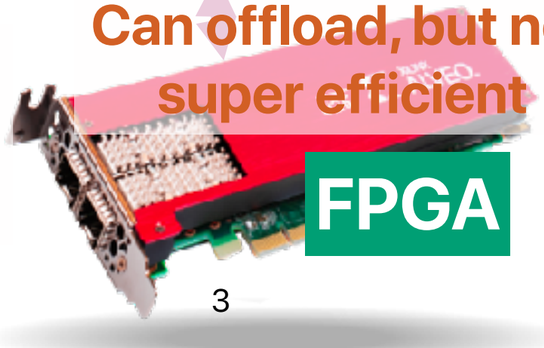
PCI EXPRESS

PCIe Root Complex

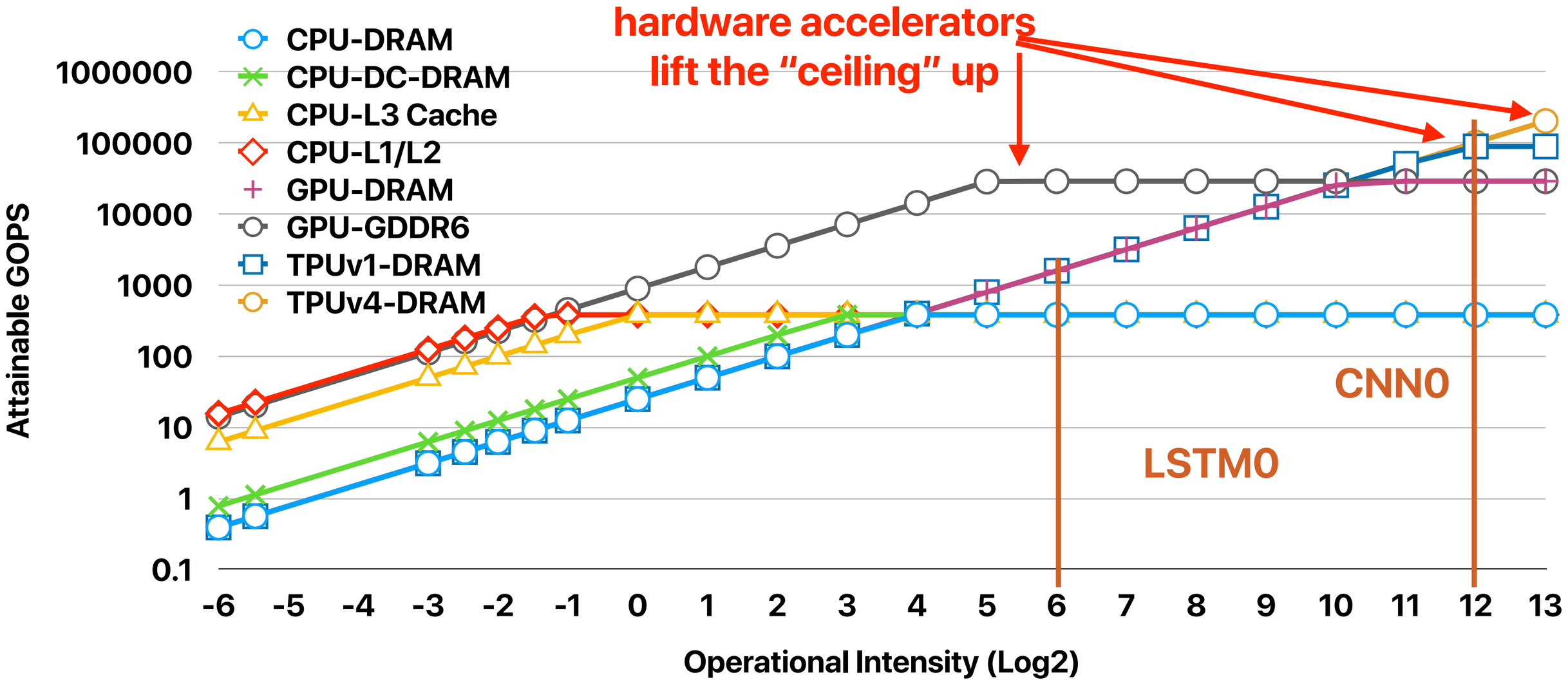


General-purpose for "regular" workloads, but can be power consuming

Can offload, but not super efficient



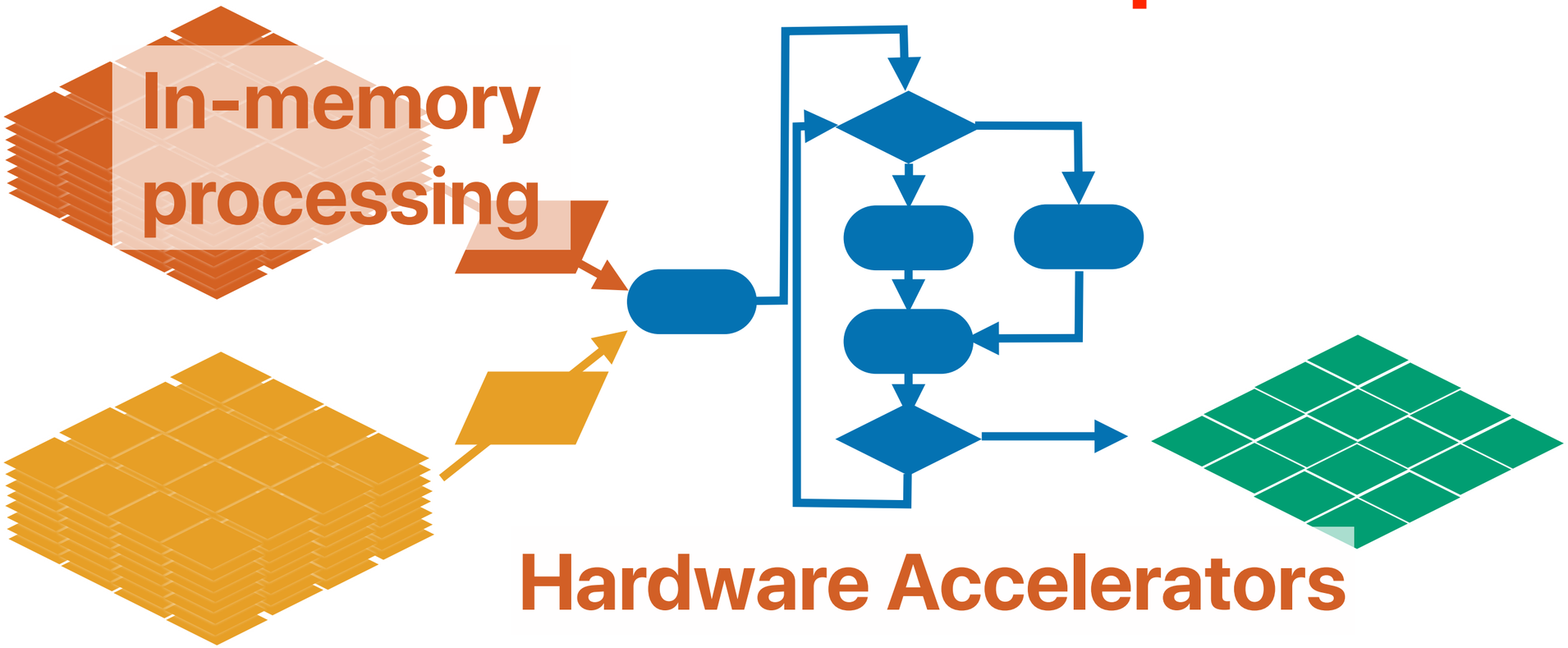
Recap: the rooflines of modern systems



Recap: what have we learned this quarter

In-Storage Processing

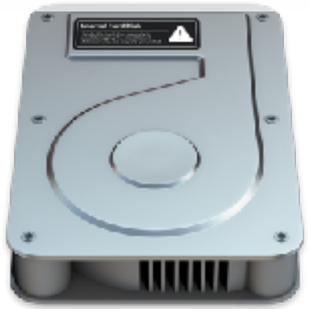
In-memory processing



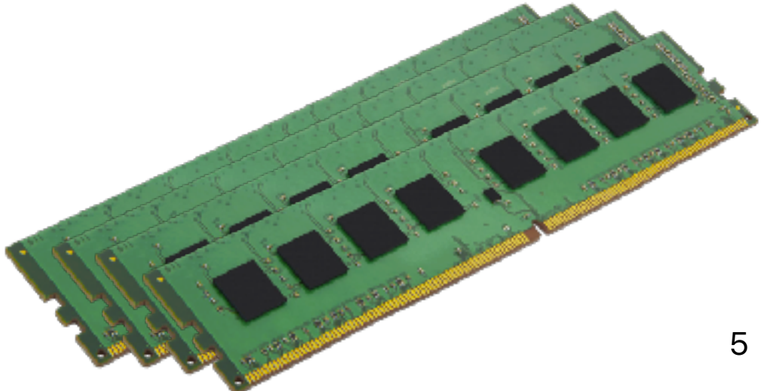
Hardware Accelerators

Right data formats

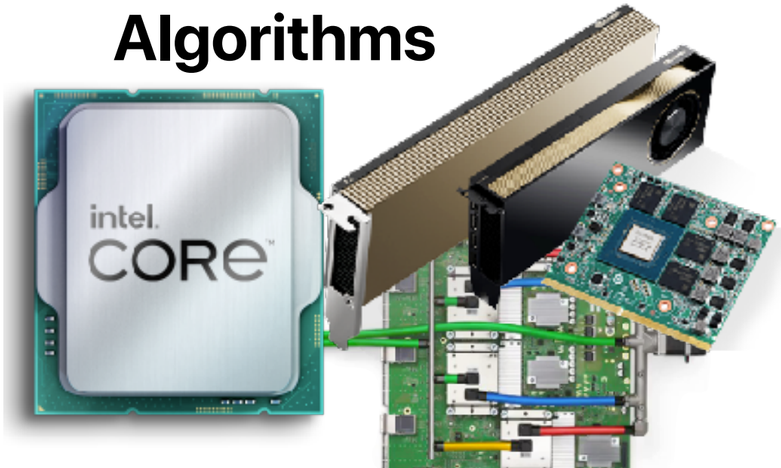
Source "data"



"Data" structures



Algorithms



Result

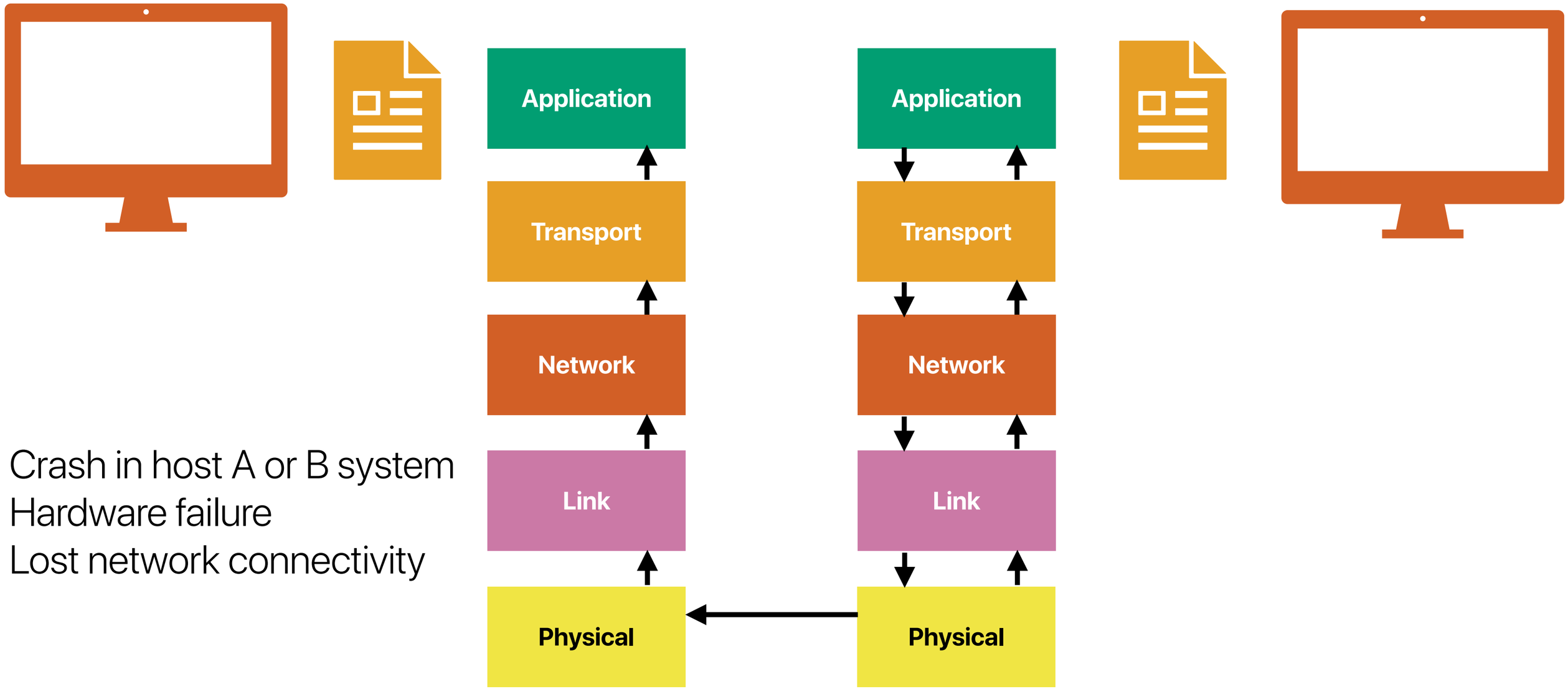
3:00

What's "the end-to-end argument"

e2e argument

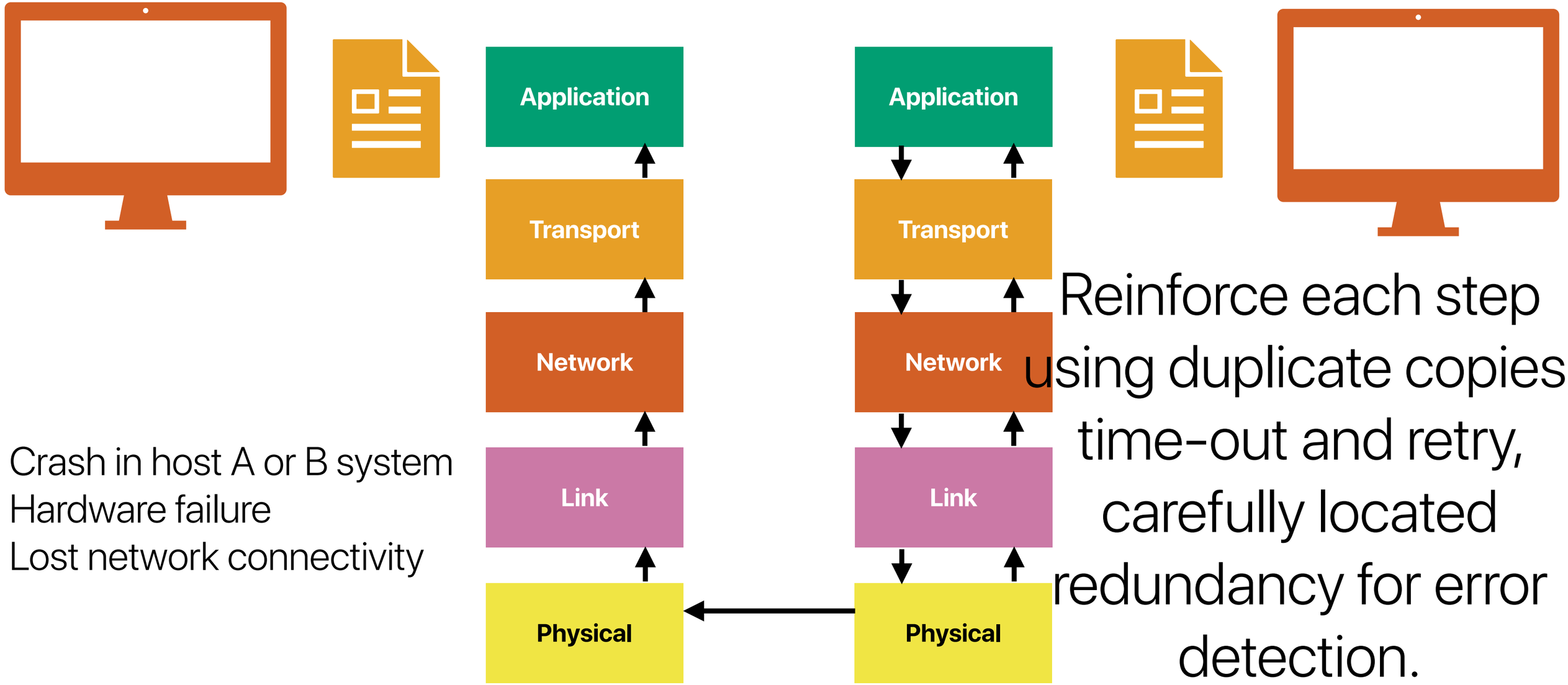
The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the endpoints of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

Example — file transfer



Crash in host A or B system
Hardware failure
Lost network connectivity

File transfer: current approach



Crash in host A or B system
Hardware failure
Lost network connectivity

Reinforce each step using duplicate copies, time-out and retry, carefully located redundancy for error detection.

e2e argument

- The application program follows the simple steps in transferring the file.
- As the final step, host B will recalculate the checksum for the transferred file.
- Then host B will send this value to be compared with the original value at host A.
- If this value doesn’t mach, the file will be resend again to host A.

Why the e2e argument?

- If we consider a network that is somewhat unreliable, dropping one packet of each hundred packet sent
- Using the above outline strategy the above network will do a bad job.
- Clearly , some effort at the lower levels to improve the network reliability will significantly improve the performance.
- The key idea here: The lower level need not to provide “perfect” reliability
- E2E check of the file transfer application must be implemented no matter how reliable the communication system become.
- The amount of effort to put into reliability within the data communication system is seen to be an engineering trade-off based on performance.

Why the e2e argument?

- Using performance to justify placing functions in low-level subsystem must be done carefully since performing the function in the low level may cost more because:
  - Some application which use the same low level subsystem and doesn’t need this function will pay for it anyway.
  - Low-level system may not have as much information as the higher level, so it cannot do the job efficiently.

Summary of e2e arguments

- All applications might not use the service
- Trade-off between performance and cost
- Combine network and application information to optimize performance

3:00

What idea(s) we've learned this quarter is/are against the e2e argument?
What idea(s) we've learned this quarter supporting the e2e argument?

Ideas against the e2e argument

Ideas against the e2e argument

- In-memory processing
  - What the e2e argument hits?
  - What the e2e argument gets wrong?
- In-storage processing
  - What the e2e argument hits?
  - What the e2e argument gets wrong?

Ideas against the e2e argument

- In-memory processing
  - What the e2e argument hits?
    - Programming efforts and inefficiency from the lack of high-level information
  - What the e2e argument gets wrong?
- In-storage processing
  - What the e2e argument hits?
    - Lack of knowledge in file systems and must consult host computers a lot
  - What the e2e argument gets wrong?

Ideas for the e2e argument

Ideas against the e2e argument

- GPGPU/GPTPU?

3:00

What have you learned from “hints for computer system design”

Hints for computer system design

Butler W. Lampson
Computer Science Laboratory Xerox Palo Alto Research Center

Hints for computer system design

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End to end Safety first	End to end
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

What ideas we've learned this quarter mapped well to this paper?

3:00

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End to end	End to end
		Safety first	
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Hints for computer system design

Why?	*Functionality* Does it work?	*Speed* Is it fast enough?	*Fault-tolerance* Does it keep working?
Where?			
Completeness	Separate normal and worst case	Shed load End to end	End to end
		Safety first	
Interface	Do one thing well: Don't generalize Get it right Don't hide power Use procedure arguments Leave it to the client Keep basic interfaces stable Keep a place to stand	Make it fast Split resources Static analysis Dynamic translation	End-to-end Log updates Make actions atomic
Implementation	Plan to throw one away Keep secrets Use a good idea again Divide and conquer	Cache answers Use hints Use brute force Compute in background Batch processing	Make actions atomic Use hints

Amdahl's Law

correctness matters

Hardware accelerators

FTL

Caching

Completeness

- Separate normal and worst case
- Make normal case fast
- The worst case must make progress
  - Saturation
  - Thrashing

Interface — Keep it simple, stupid

- Do one thing at a time or do it well
  - Don’t generalize
  - Example
    - Interlisp-D stores each virtual page on a dedicated disk page
      - 900 lines of code for files, 500 lines of code for paging
      - fast — page fault needs one disk access, constant computing cost
    - Pilot system allows virtual pages to be mapped to file pages
      - 11000 lines of code
      - Slower — two disk accesses in handling a page fault, under utilize the disk speed
- Get it right

More on Interfaces

- Make it fast, rather than general or powerful
  - CISC v.s. RISC
- Don’t hide power
  - Are we doing all right with FTL?
- Use procedure arguments to provide flexibility in an interface
  - Thinking about SQL v.s. function calls
- Leave it to the client
  - Monitors’ scheduling
  - Unix’s I/O streams

Implementation

- Keep basic interfaces stable
  - What happen if you changed something in the header file?
- Keep a place to stand if you do have to change interfaces
  - Mach/Sprite are both compatible with existing UNIX even though they completely rewrote the kernel
- Plan to throw one away
- Keep secrets of the implementation — make no assumption other system components
  - Don’t assume you will definitely have less than 16K objects!
- Use a good idea again
  - Caching!
  - Replicas
- Divide and conquer

Speed

- Split resources in a fixed way if in doubt, rather than sharing them
  - Processes
  - VMM: Multiplexing resources Guest OSs aren't even aware that they're sharing
- Use static analysis — compilers
- Dynamic translation from a convenient (compact, easily modified or easily displayed) representation to one that can be quickly interpreted is an important variation on the old idea of compiling
  - Java byte-code
  - LLVM
- Cache answers to expensive computations, rather than doing them over
- Use hints to speed up normal execution
  - The Ethernet: carrier sensing, exponential backoff

Speed

- When in doubt, use brute force
- Compute in background when possible
  - Free list instead of swapping out on demand
  - Cleanup in log structured file systems: segment cleaning could be scheduled at nighttime.
- Use batch processing if possible
  - Soft timers: uses trigger states to batch process handling events to avoid trashing the cache more often than necessary
  - Write buffers
- Safety first
- Shed load to control demand, rather than allowing the system to become overloaded
  - Thread pool
  - MLQ scheduling
  - Working set algorithm
  - Xen v.s. VMWare

Electrical
Computer Science
Engineering

277

つづく