What kind of distributed system is a multicore machine?

Stefan Kaestle
stefan.kaestle@inf.ethz.ch

Systems Group @ ETH Zurich

Who am I?

Systems@ETH Zürich

- PhD student with Prof. Timothy Roscoe
- Working on operating systems (Barrelfish)
  - But this talk is not only about that

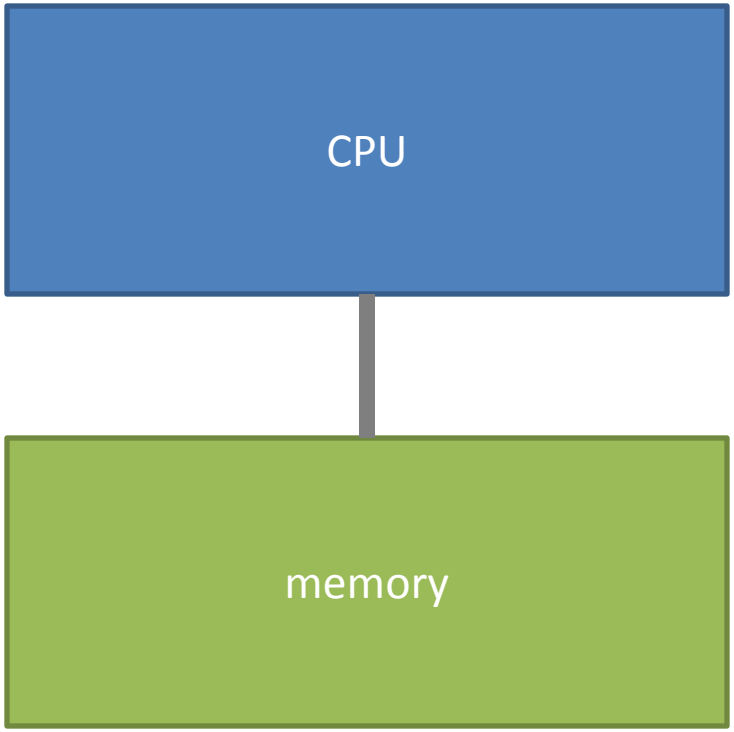
- I will present
  - **Trends** of multicore hardware
  - Ongoing **research** in the Systems Group
  - Also: **Opportunities** for future research

Some of it is preliminary work. Lots of unknowns, **feedback welcome**

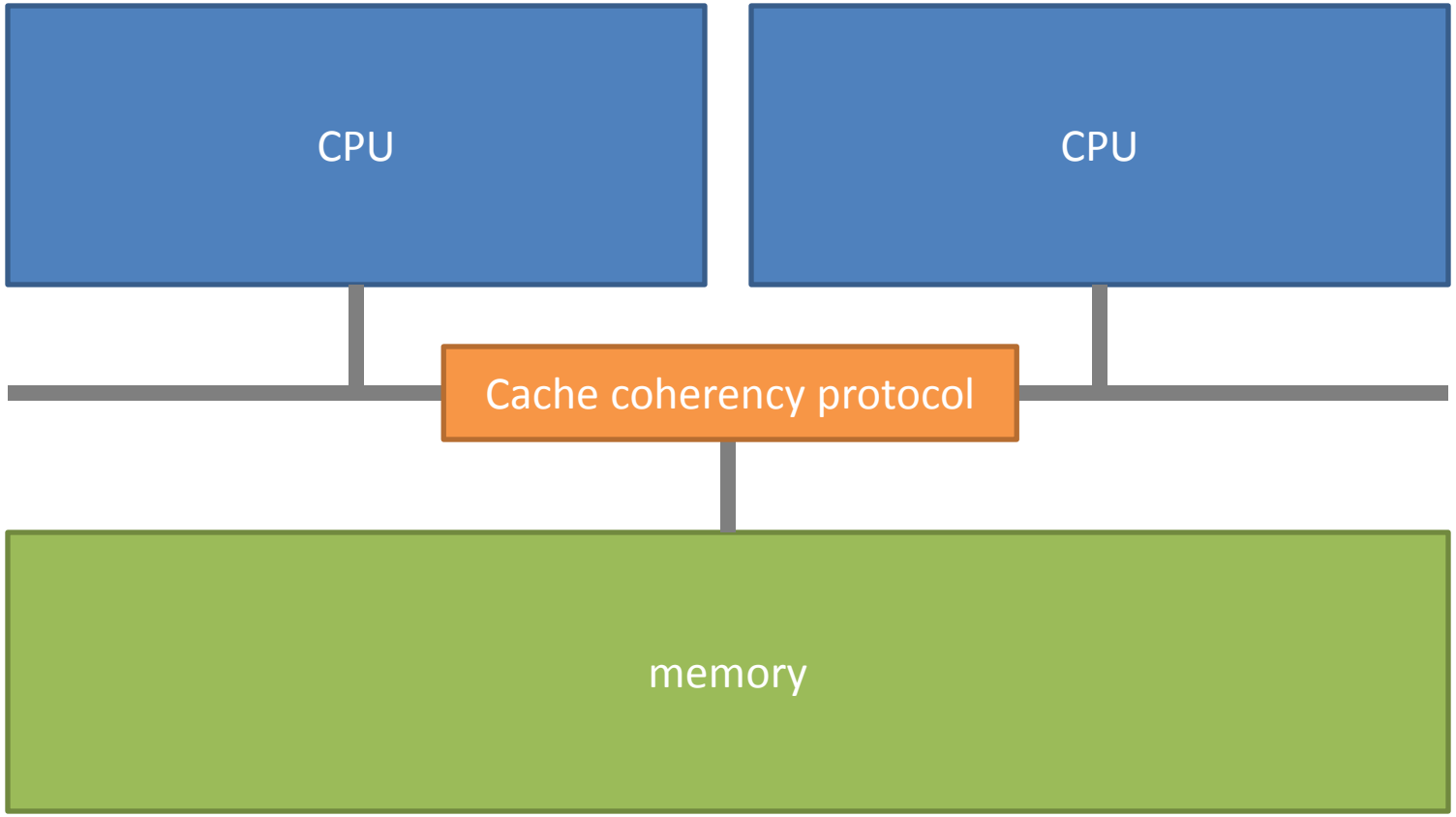
Systems@ETH Zürich

MULTICORE INTRODUCTION

Computer 20 years ago

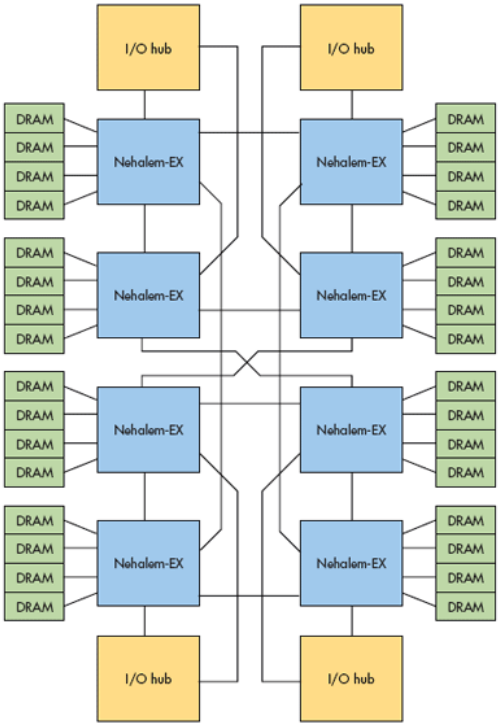


Computer 10 years ago



Today

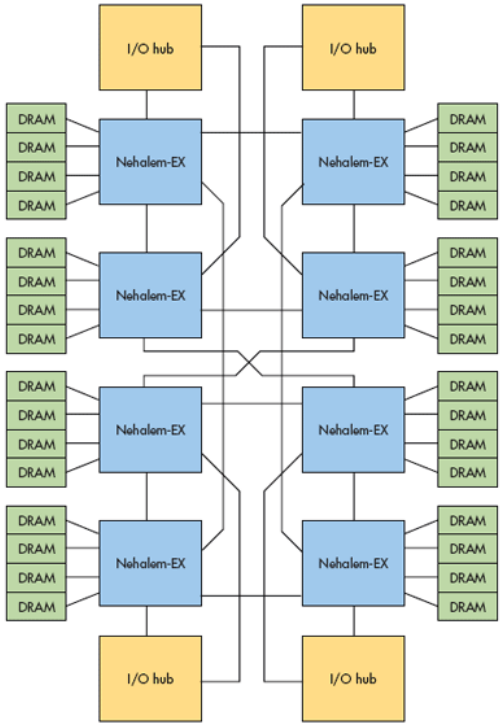
- Multicores:
  - Increasing number of cores
  - NUMA nodes
    - Local memory controllers
    - shared resources
  - Interconnect (not exposed)



So what is this talk about?
aka why am I here?

Systems@ETH Zürich

- Multicores:
  - Increasing number of cores
  - NUMA nodes
    - Local memory controllers, shared resources
  - Interconnect (not exposed)

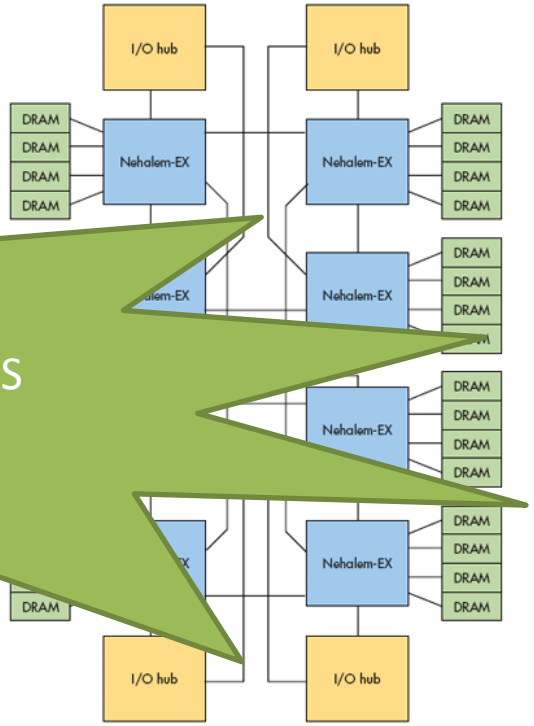


→ Looks like **distributed systems**

So what is this talk about?
aka why am I here?

Systems@ETH Zürich

- Multicores:
  - Increasing num[illegible] of cores
  - – NUM[illegible]
  - – Intercon[illegible]



Oh great, so lets just apply traditional DS algorithms

→ Looks like **distributed systems**

Systems@ETH Zürich

Example: replication of data

PROGRAM AS DISTRIBUTED SYSTEM

Interconnect characteristics

Systems@ETH Zürich

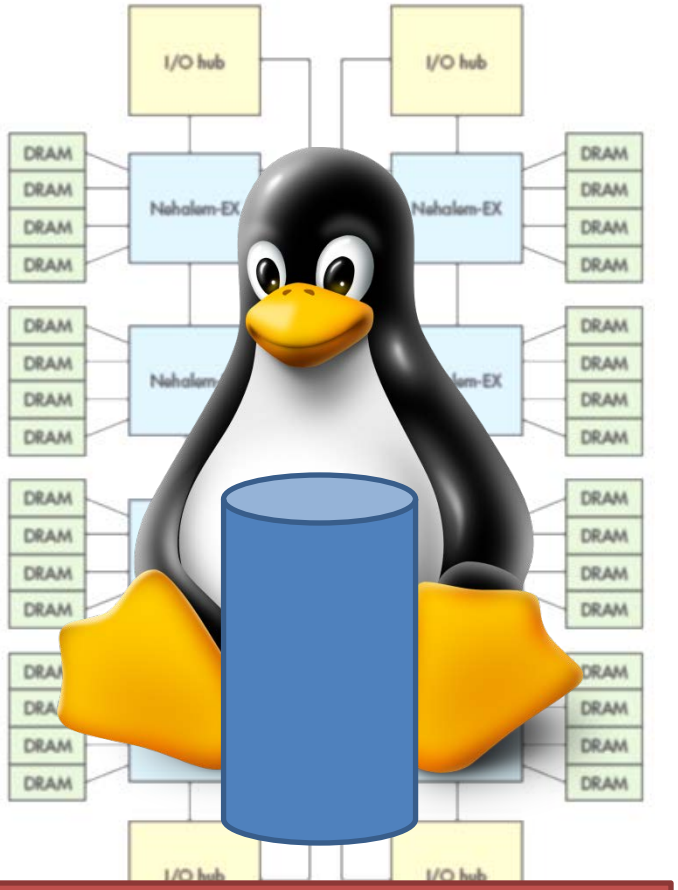
In common:

- **Congestion**
- **Package** based (internally)
- Routing

Systems@ETH Zürich

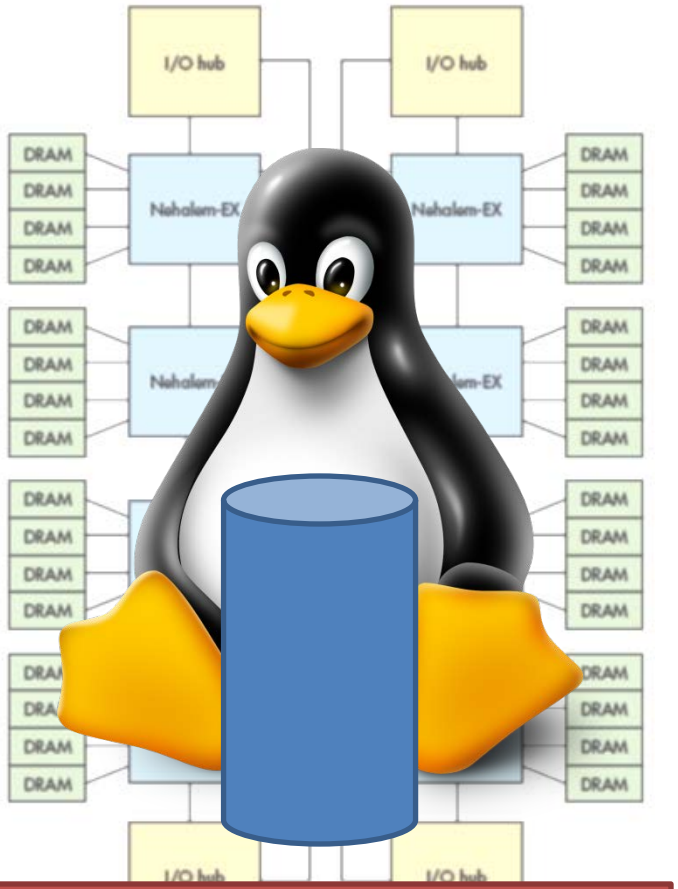
SHARED NOTHING ARCHITECTURE

Multikernel OS (Barrelfish)

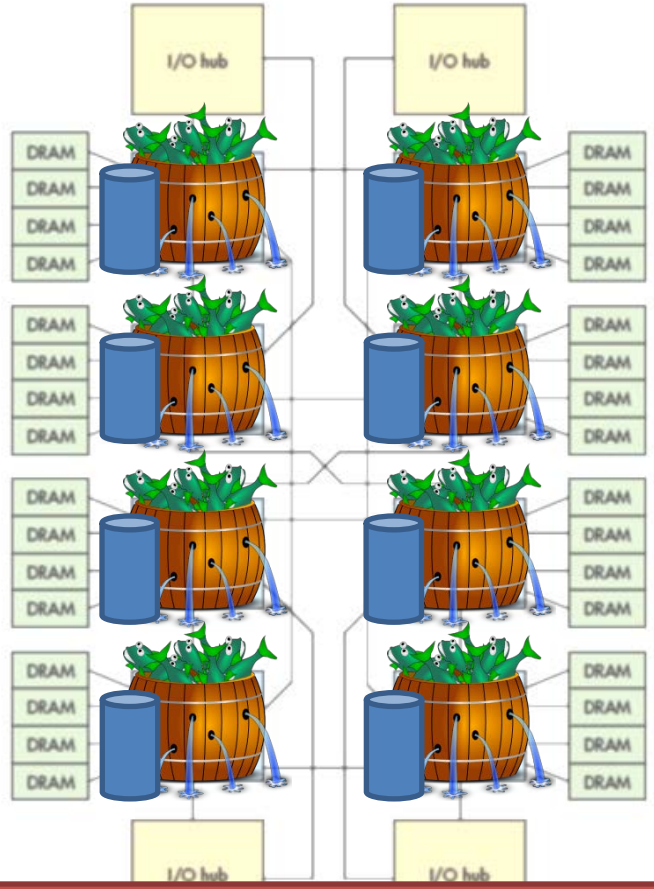


Shared memory programming

Multikernel OS (Barrelfish)



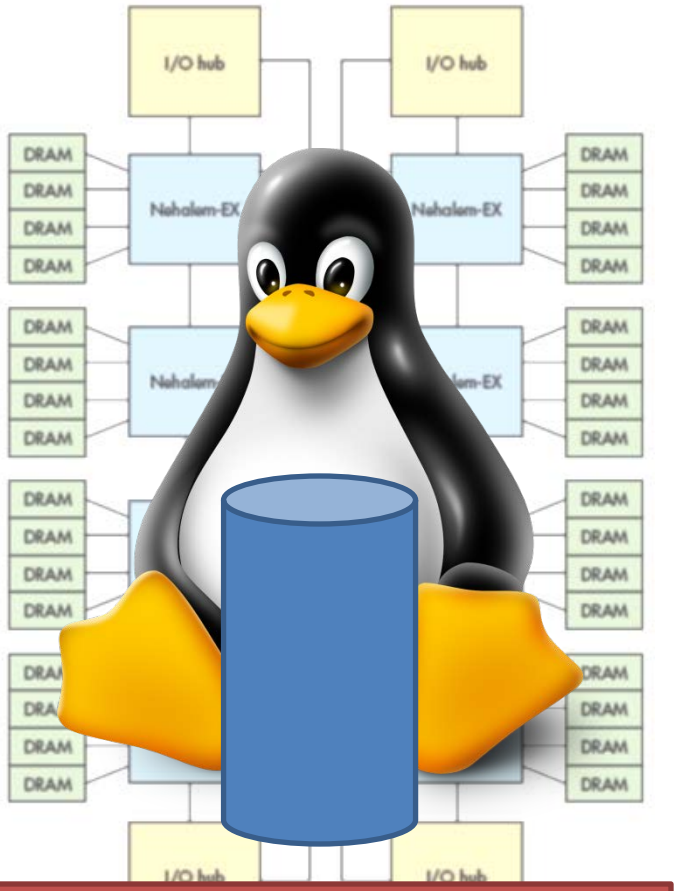
Shared memory programming



Message passing

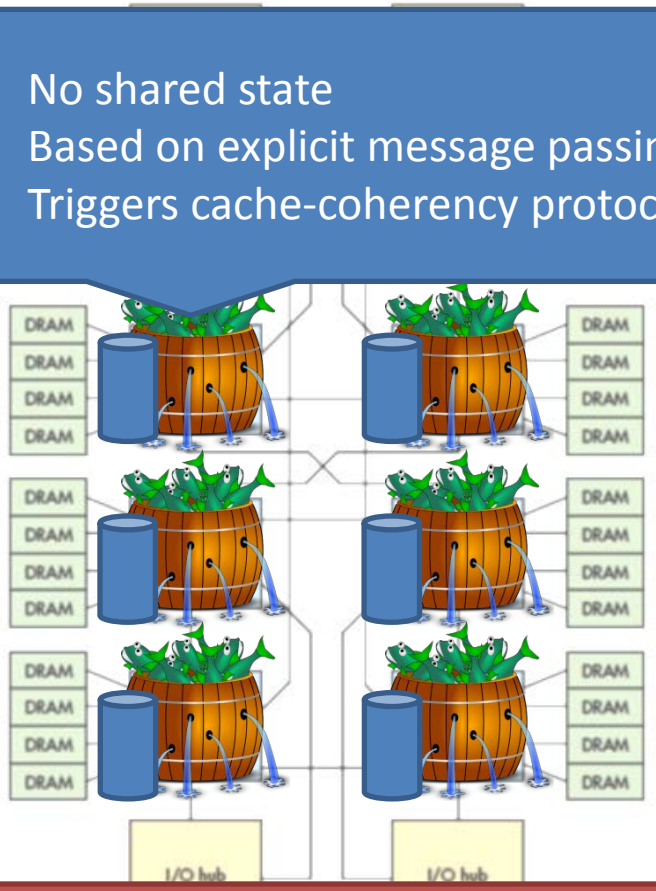
Multikernel OS (Barrelfish)

Systems@ETH Zürich



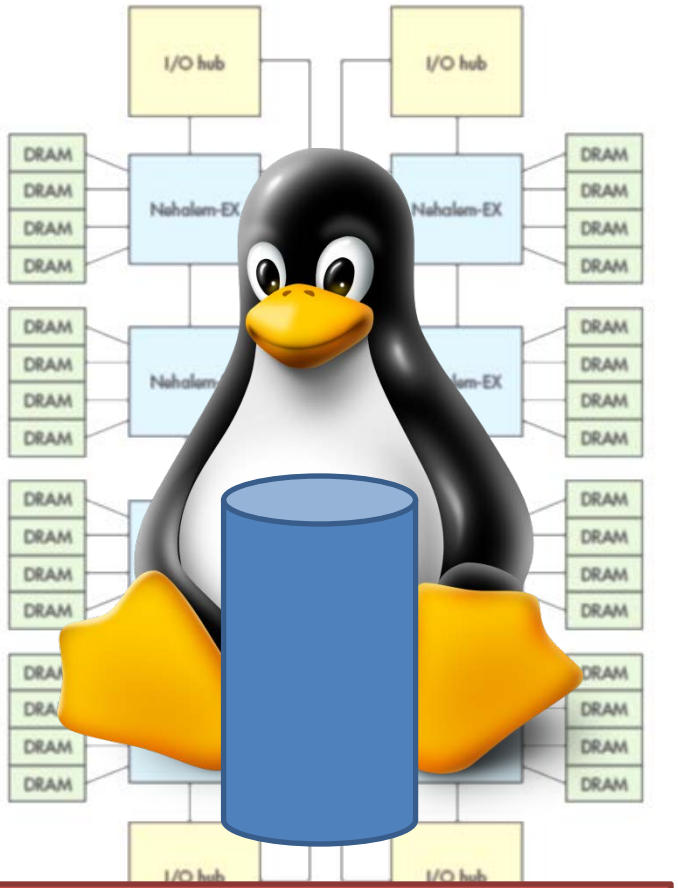
Shared memory programming

- No shared state
- Based on explicit message passing
- Triggers cache-coherency protocol



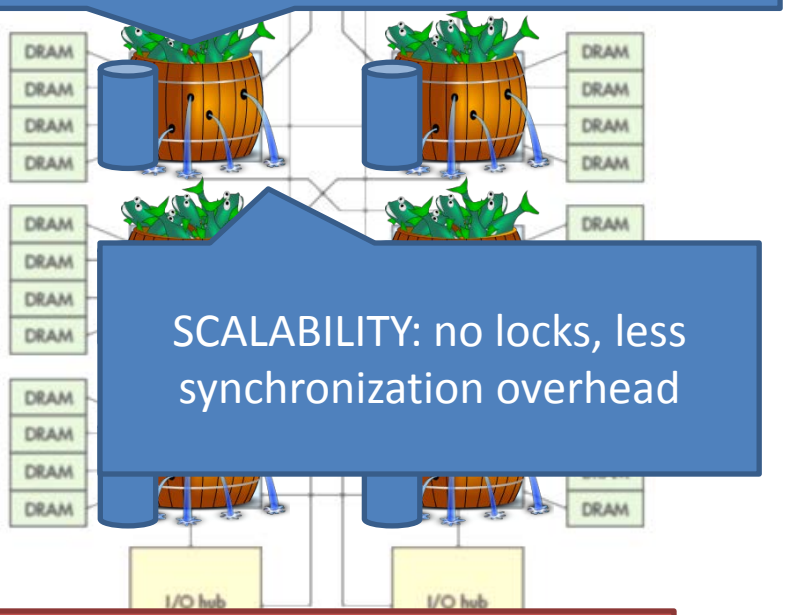
Message passing

Multikernel OS (Barrelfish)



Shared memory programming

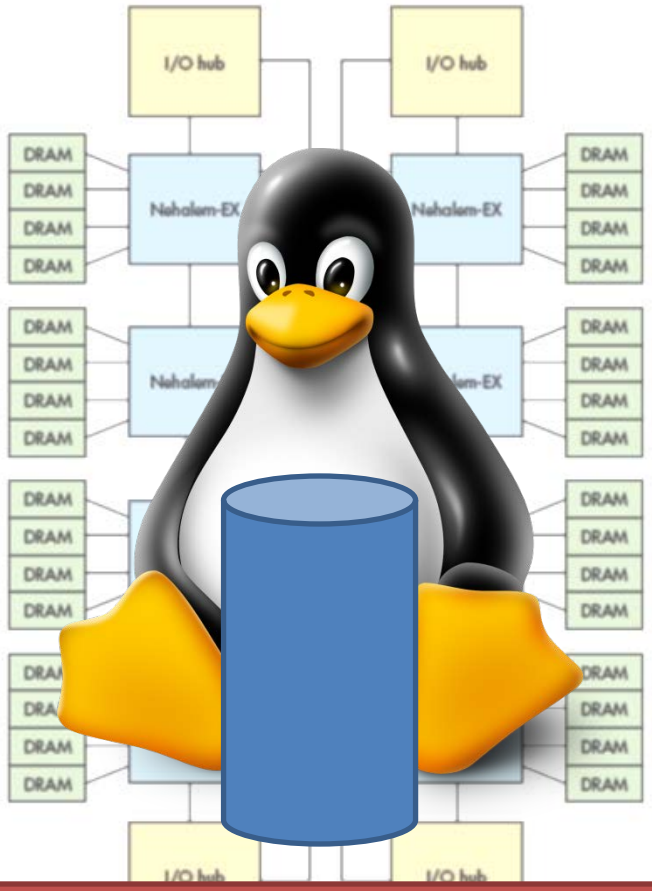
- No shared state
- Based on explicit message passing
- Triggers cache-coherency protocol



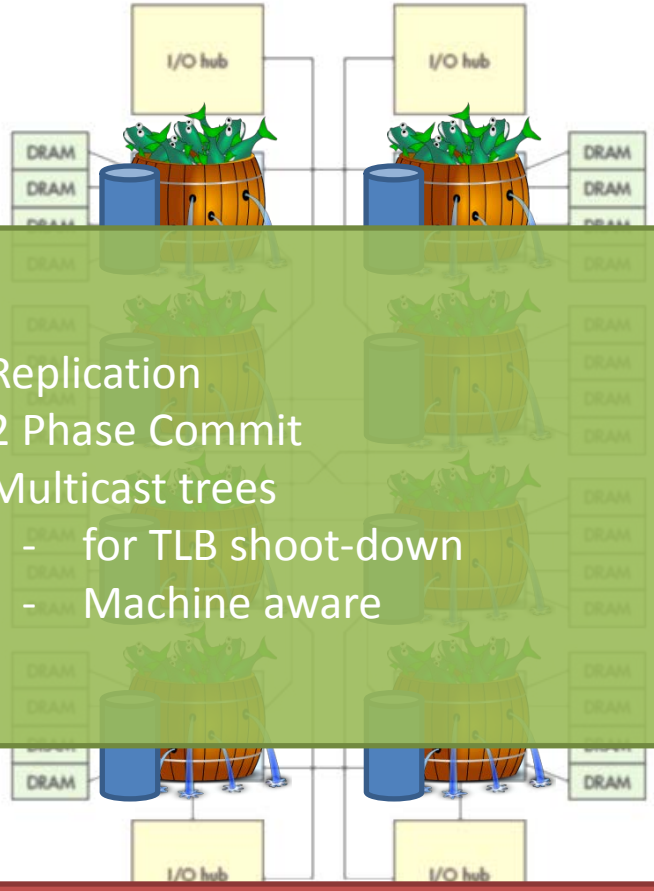
SCALABILITY: no locks, less synchronization overhead

Message passing

Multikernel OS (Barrelfish)



Shared memory programming



- Replication
- 2 Phase Commit
- Multicast trees
  - for TLB shoot-down
  - Machine aware

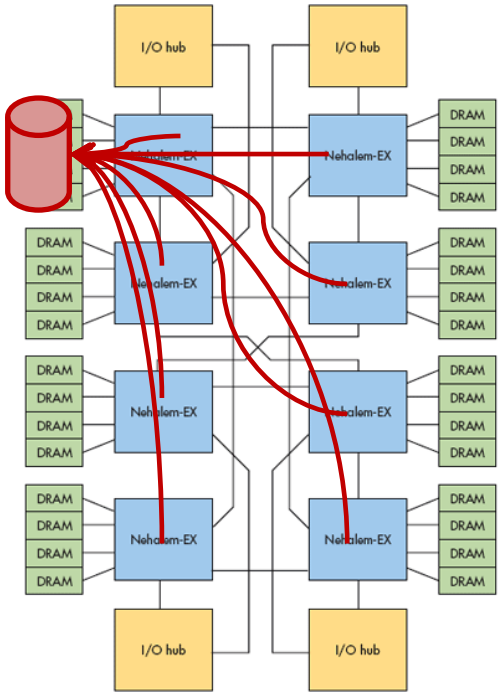
Message passing

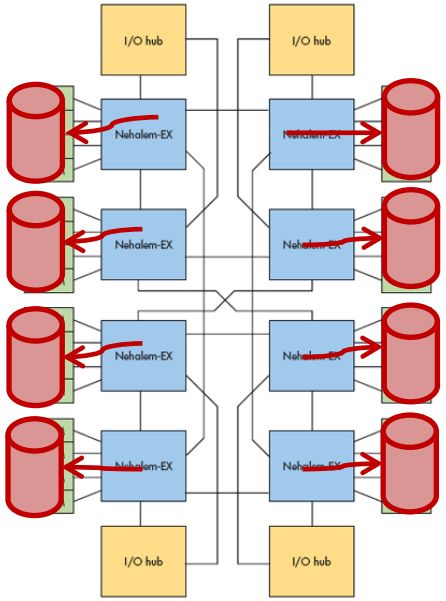
Reduce interconnect traffic

Interconnect congestion (Shoal)

Systems@ETH Zürich

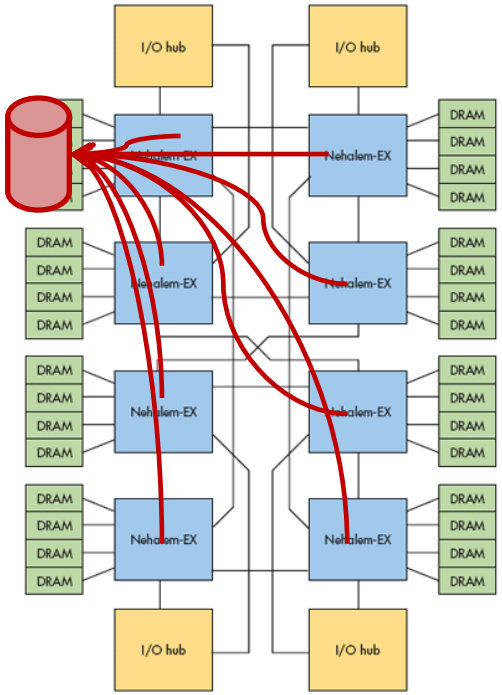
- Bad memory allocation
- Replication/distribution



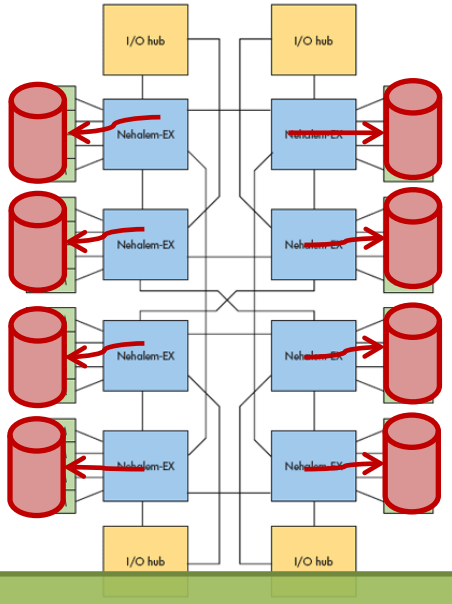


Interconnect congestion (Shoal)

- Bad memory allocation



- Replication/distribution

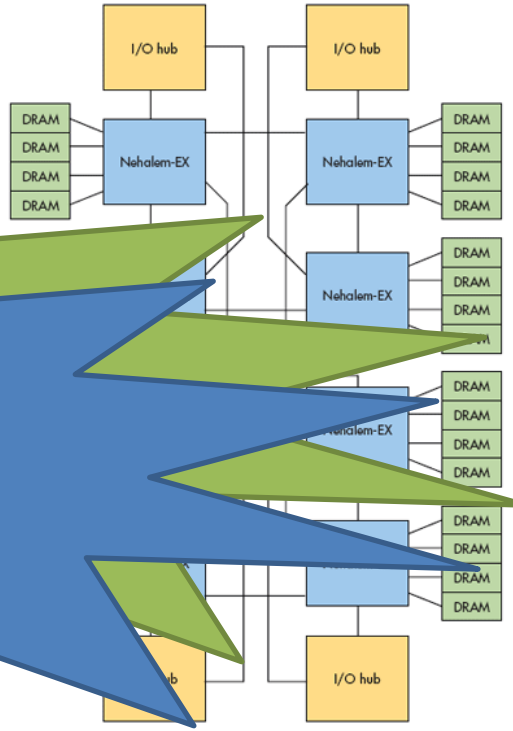


- Reduces traffic on interconnect

So what is this talk about?
aka why am I here?

Systems@ETH Zürich

- Multicores:
  - Increasing number of cores
  - [illegible]



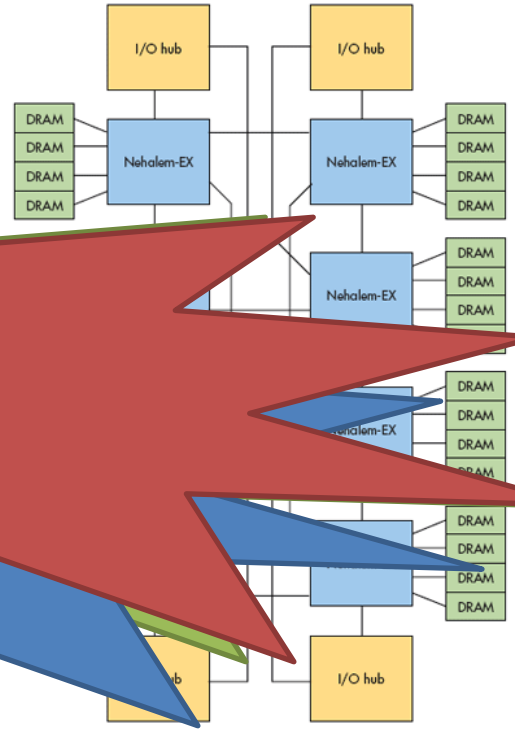
Okay, so all good? Can we go home now?

→ Looks like **distributed systems**

So what is this talk about?
aka why am I here?

Systems@ETH Zürich

- Multicores:
  - Increasing number of cores
  - [illegible]



Well, not quite ..

→ Looks like **distributed systems**

DIFFERENCES TO TRADITIONAL DS

Interconnect characteristics

Systems@ETH Zürich

In common:

- **Congestion**
- **Package** based (internally)
- Routing

Differences:

- **Complexity measures**
- Reliable
- synchronous?
- Static (within a machine)
- Very concrete
- Diversity
- **Hierarchical**
- Hybrid

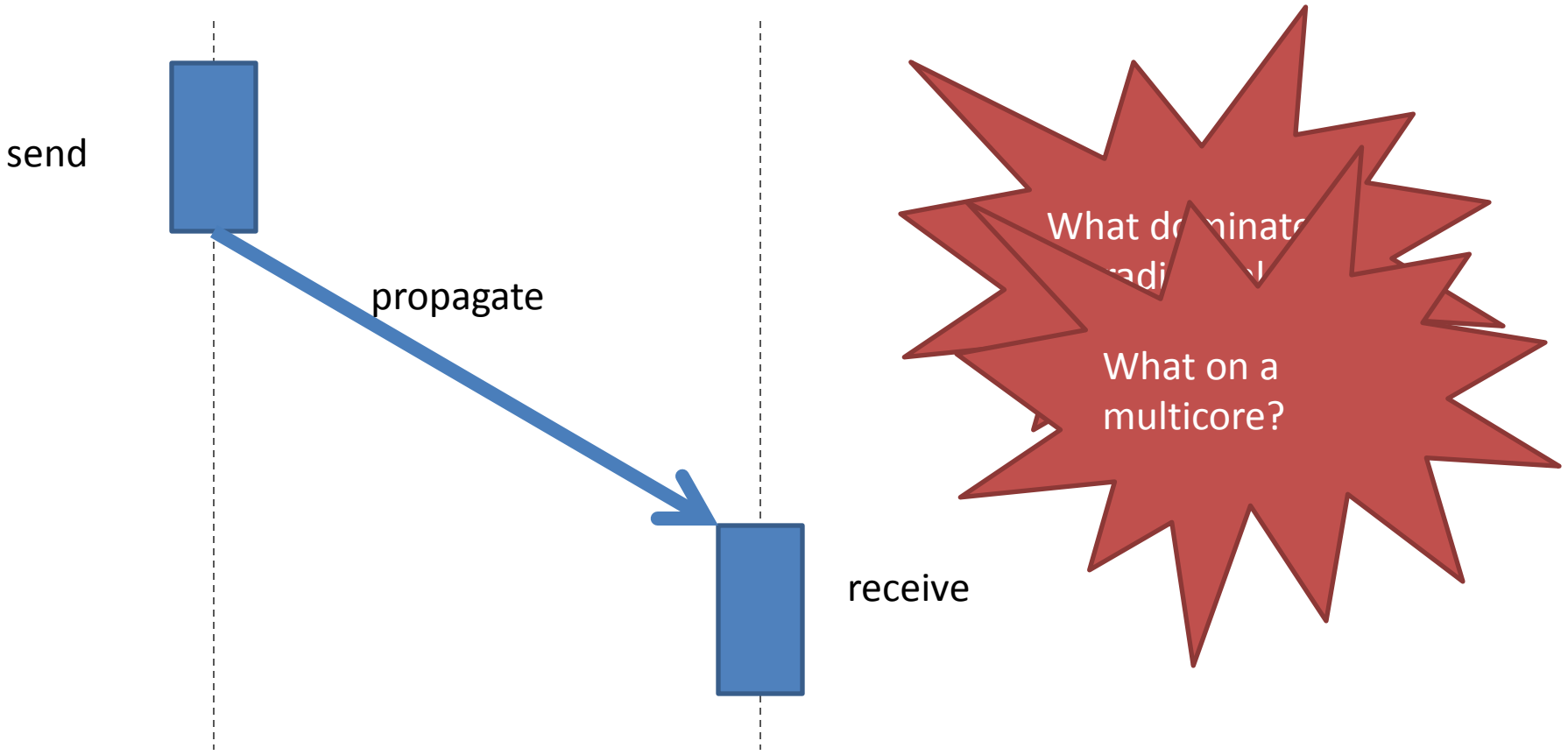
And many more ..

Systems@ETH Zürich

Complexity metrics

DIFFERENCES: AN EXAMPLE

Complexity metrics



Complexity metrics

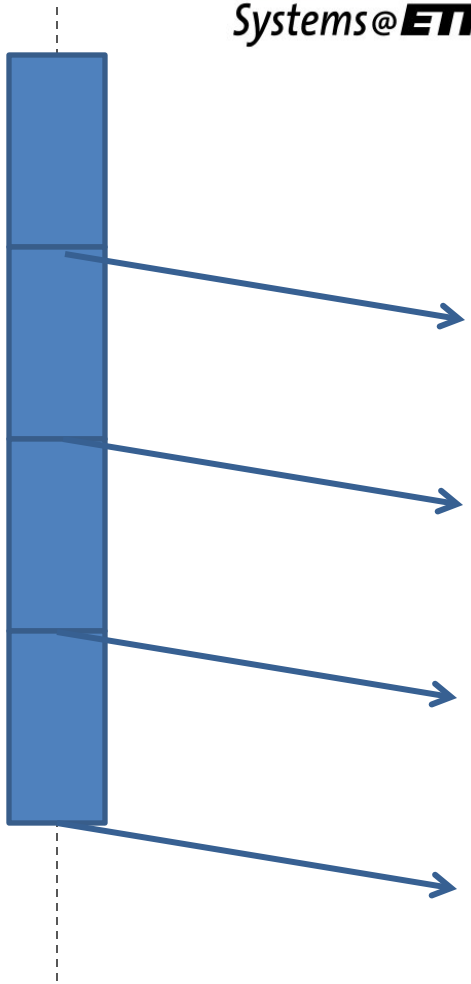
Systems@ETH Zürich

- Traditionally:
  - propagation time dominates
  - **#rounds** (#messages/round irrelevant)
- Multicore:
  - Propagation cheap
  - **Send and receive expensive**
    - Interrupts, device driver communication, multiplexing, (un-) marshaling, scheduling

Example: broadcast

- Broadcast to n clients:

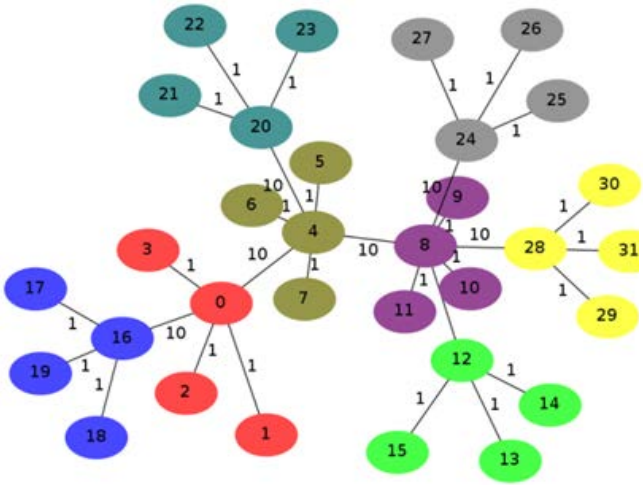
- Traditionally: send sequentially
- Multicore: BAD
  - `cost(seq): O(n)`

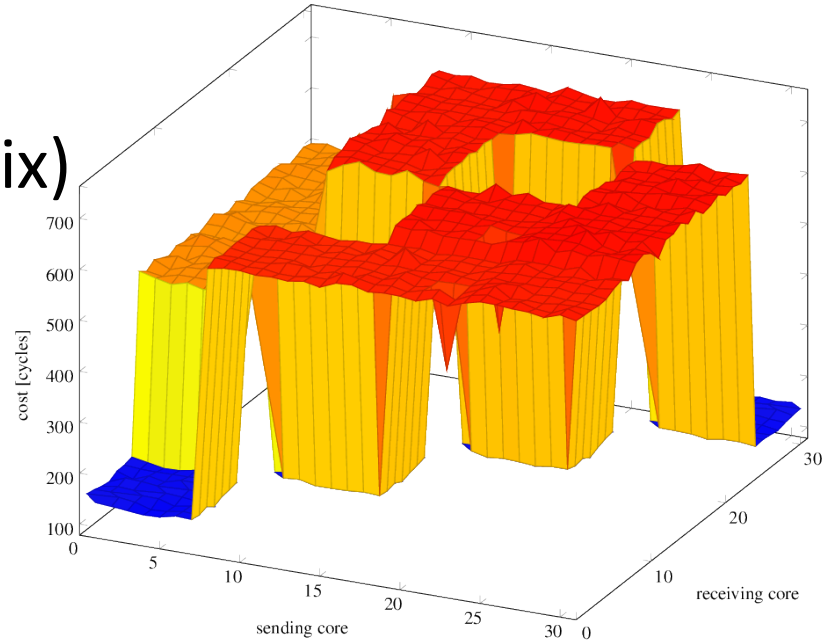


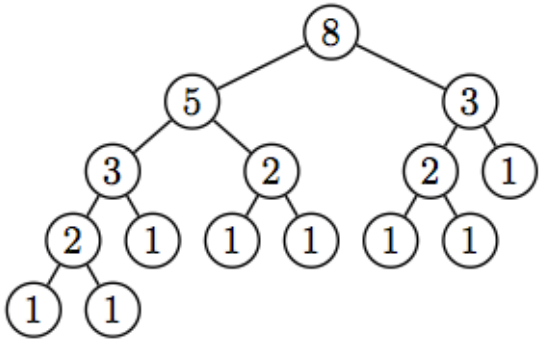
Multicore/Broadcast

→ Tree, **NOT** balanced

(ideally: topology-aware, Radix)







Multicore/Broadcast++

Systems@ETH Zürich

- **Leverage shared resources**
- **Hybrid** algorithm:
  - Message passing across nodes
  - Shared memory inside of nodes

- Compose algorithm at runtime
  - machine-aware
  - scheduling-aware

Conclusions

- Multicores look like traditional DS
  - Apply ideas from DS
- But behave **differently**
  - Need to **re-evaluate** distributed algorithms

Systems@ETH Zürich

Failure Model

DIFFERENCES

Consensus

Systems@ETH Zürich

- RAFT/Paxos
- Need to reduce number of messages
- Treat some clusters of cores as failure-domain
  - Allows to use weaker algorithms inside

→Compose algorithms

Failure model: **TODAY**

Systems@ETH Zürich

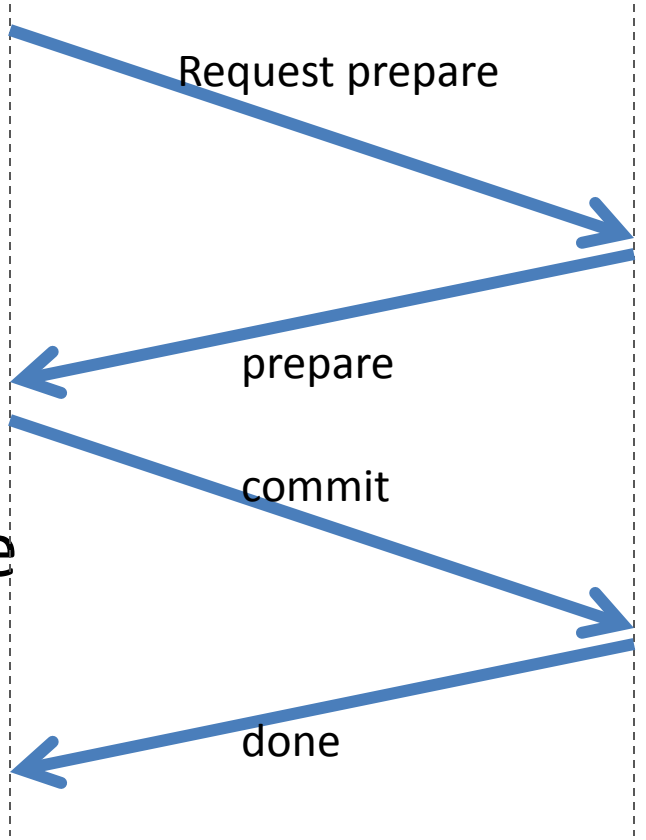
- Today: machine is reliable
  - Interconnect
    - Messages do not get lost
    - Upper bound on propagation time (synchronous)

Consensus: 2PC

- 2 Phases (1 RTT each)
  - Prepare
  - Commit

- Interconnect reliable

→ No ACKs in Commit Phase



Consensus: Paxos

Systems@ETH Zürich

- Do we want Paxos?
  - probably not, sends too many messages
- But what then?
  - Ongoing research, e.g. 1Paxos (EPFL, claims to be multicore aware)
- Failure domians?

Failure model: near **FUTURE**

Systems@ETH Zürich

- Parts of the machine can fail
  - Industry is very interested in this

- But: what is the unit of failure?
  - Parts of the machine can be treated as one failure domain (e.g. because the share resources)
    → again: hierarchy