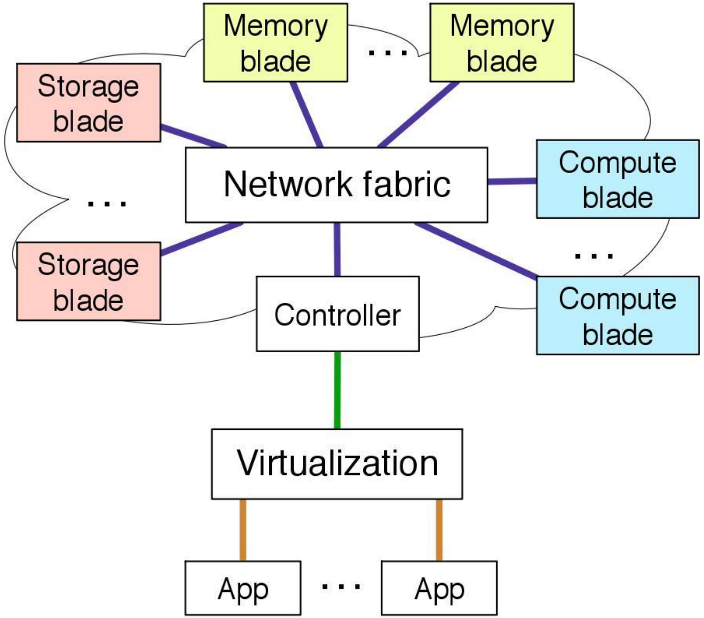
Tolerating Faults in Disaggregated Datacenters

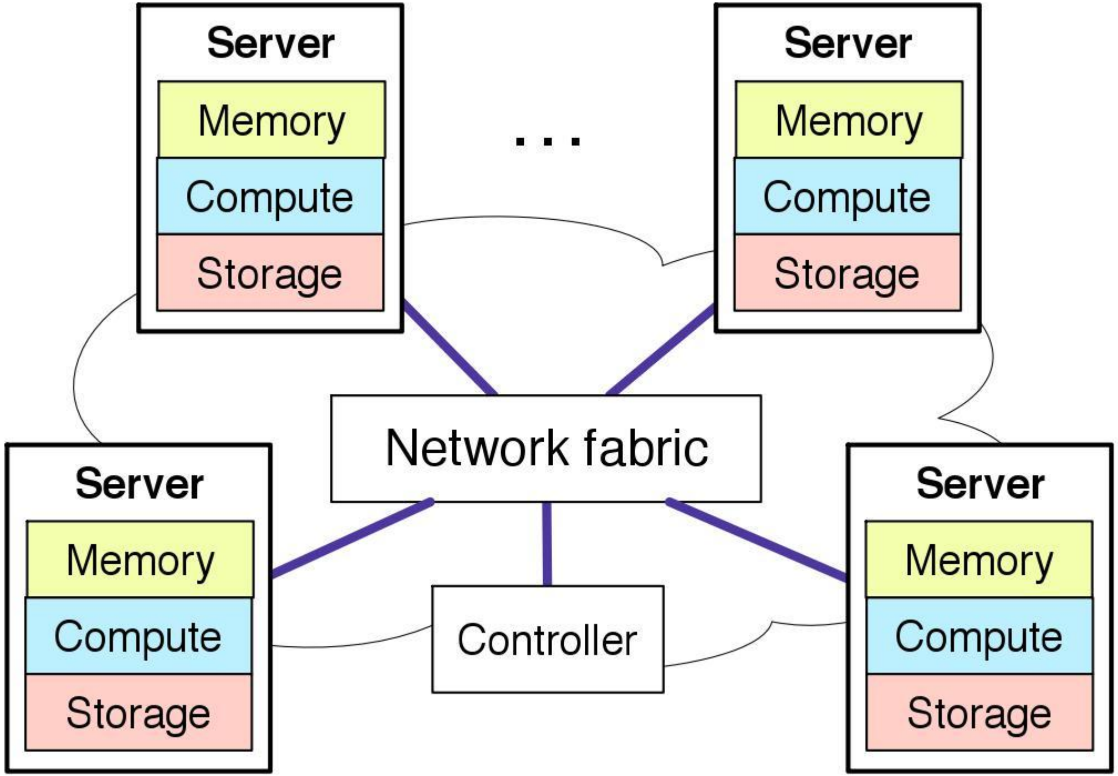


Amanda Carbonari, Ivan Beschastnikh

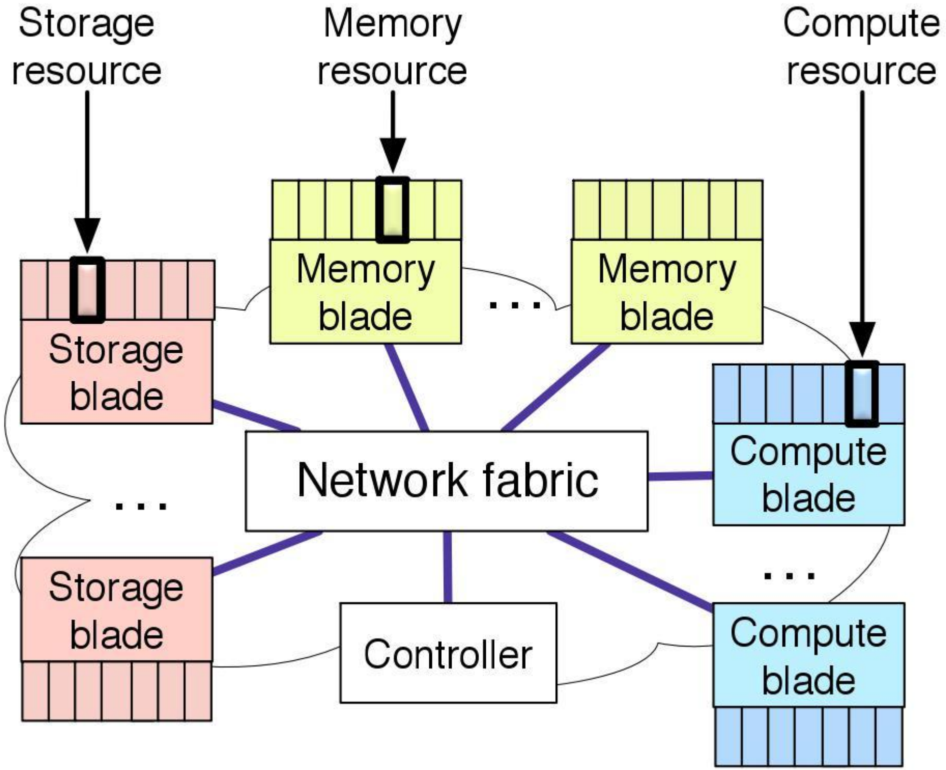
University of British Columbia

HotNets17

Today's Datacenters

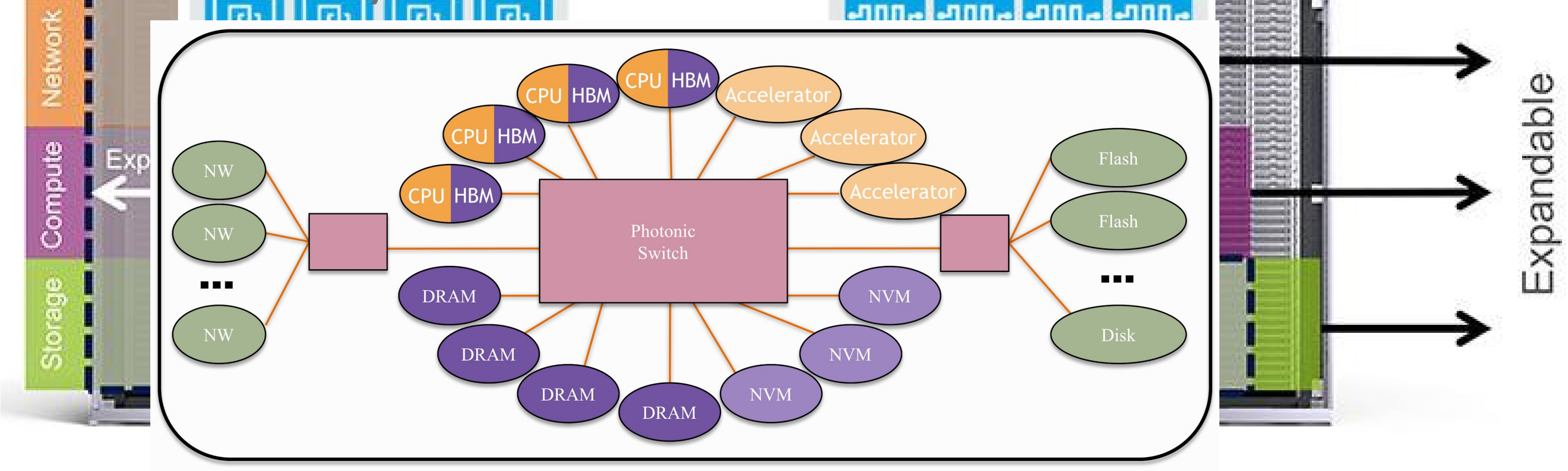


The future: Disaggregation



~~The future:~~ Disaggregation is coming

- Intel Rack Scale Design, Ericsson Hyperscale Datacenter System 8000
- HP The Machine
- UC Berkeley Firebox

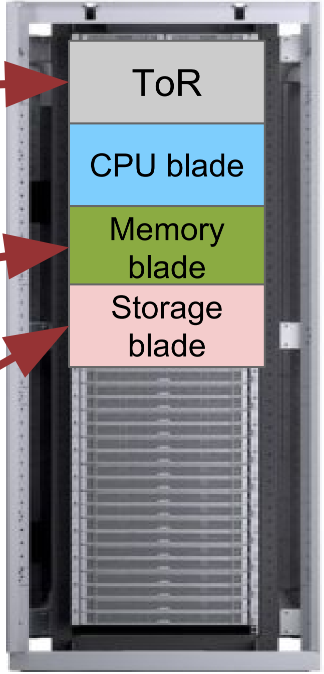


Disaggregation Research Space

Network + disaggregation [R2C2 SIGCOMM'15, Gao et. al. OSDI'16]

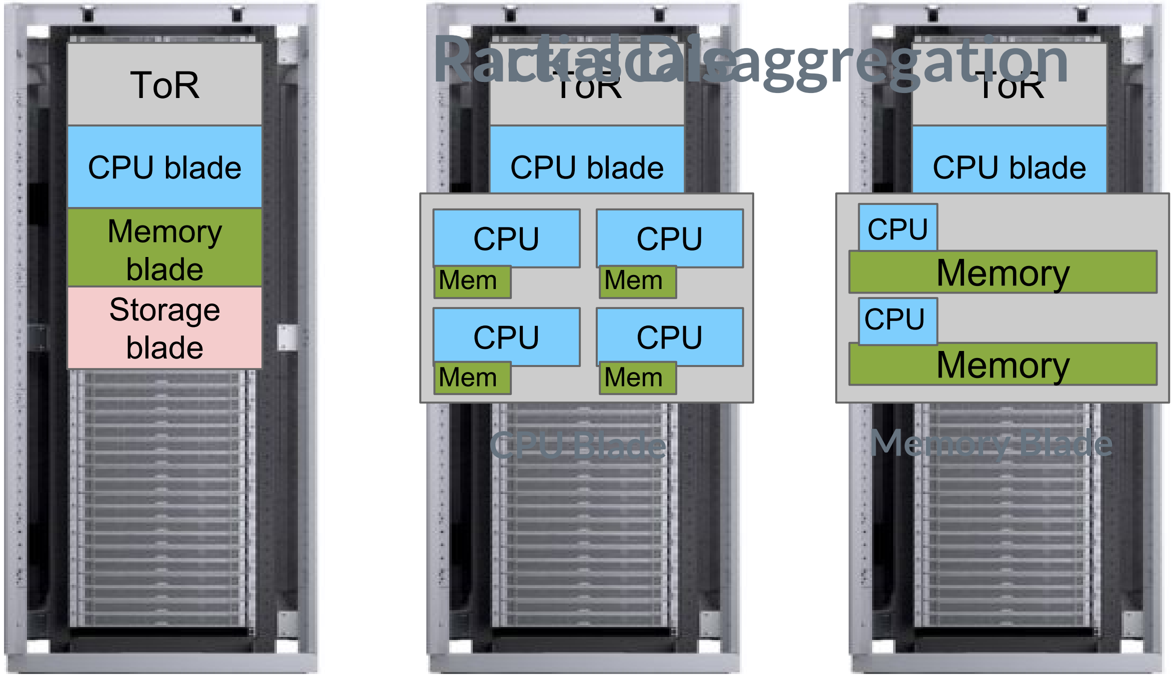
Memory disaggregation [Rao et. al. ANCS'16, Gu et. al. NSDI'17, Aguilera et. al. SoCC'17]

Flash/Storage disaggregation [Klimovic et. al. EuroSys'16, Legtchenko et. al. HotStorage'17, Decibel NSDI'17]



Our research focus: how to build systems on DDCs

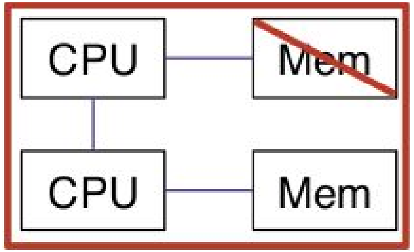
Our Assumptions



What happens if a resource fails?

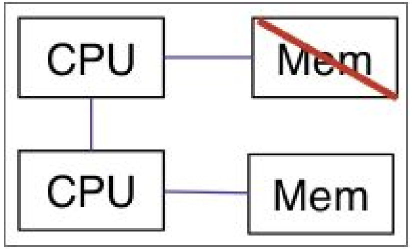
How should applications observe resource failures?

DC: resources *fate share*



Server

DDC: resources do **not** fate share



Disaggregated Server

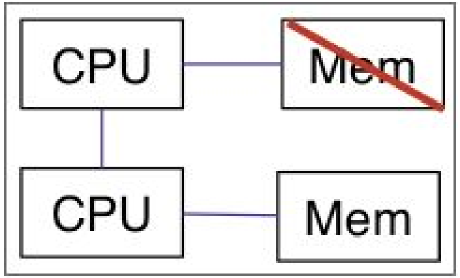
DDC fate sharing should be **enforced in the network**.

Why enforce fate sharing in the network?

- Reasonable to assume legacy applications will run on DDCs **unmodified**
- All memory accesses are across the rack network
- **Interposition layer** = Software Defined Networking (SDN)

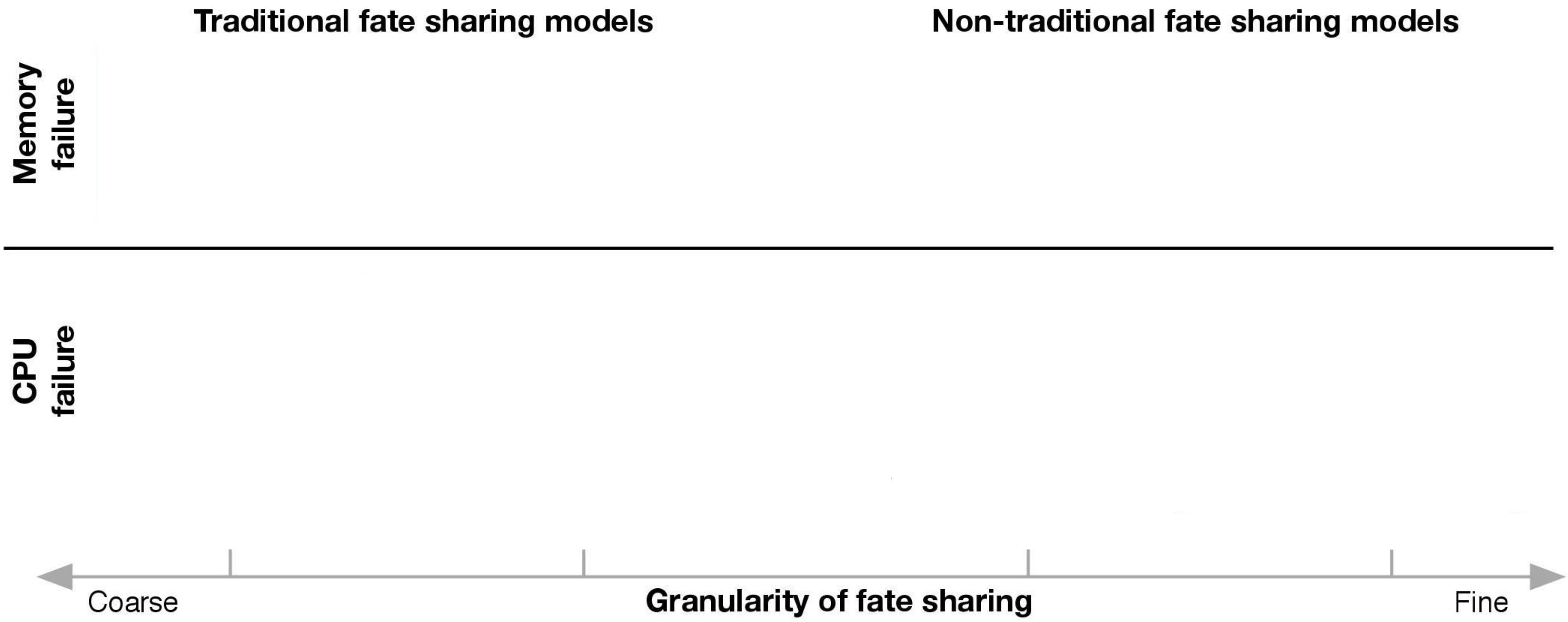
Fault tolerance in DDCs

- Fate sharing exposes a failure type to higher layers (**failure granularity**)
- Techniques inspired by related work
  - Distributed systems [Bonvin et. al. SoCC'10, GFS OSDI'03, Shen et. al. VLDB'14, Xu et. al. ICDE'16]
  - HA VMs and systems [Bressoud et. al. SOSP'95, Bernick et. al. DSN'05, Remus NSDI'08]
  - HPC [Bronevetsky et. al. PPoPP'03, Egwutuoha et. al. Journal of Supercomputing'13]



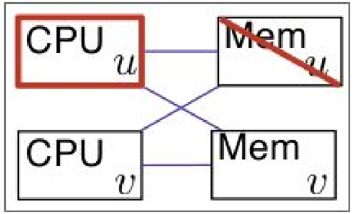
- **Open research question:** how to integrate existing fault tolerance techniques into DDC?

Fate Sharing Granularities

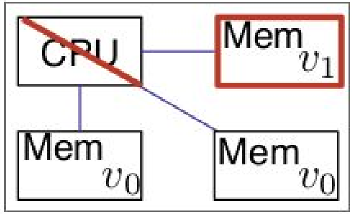


Tainted Fate Sharing

- Memory fails → CPU reading/using memory fails with
- CPU fails while writing to one replica→ inconsistent memory fails (v_1)
- Modularity vs. performance
- **Open research question:** implications of dynamic computation in-network

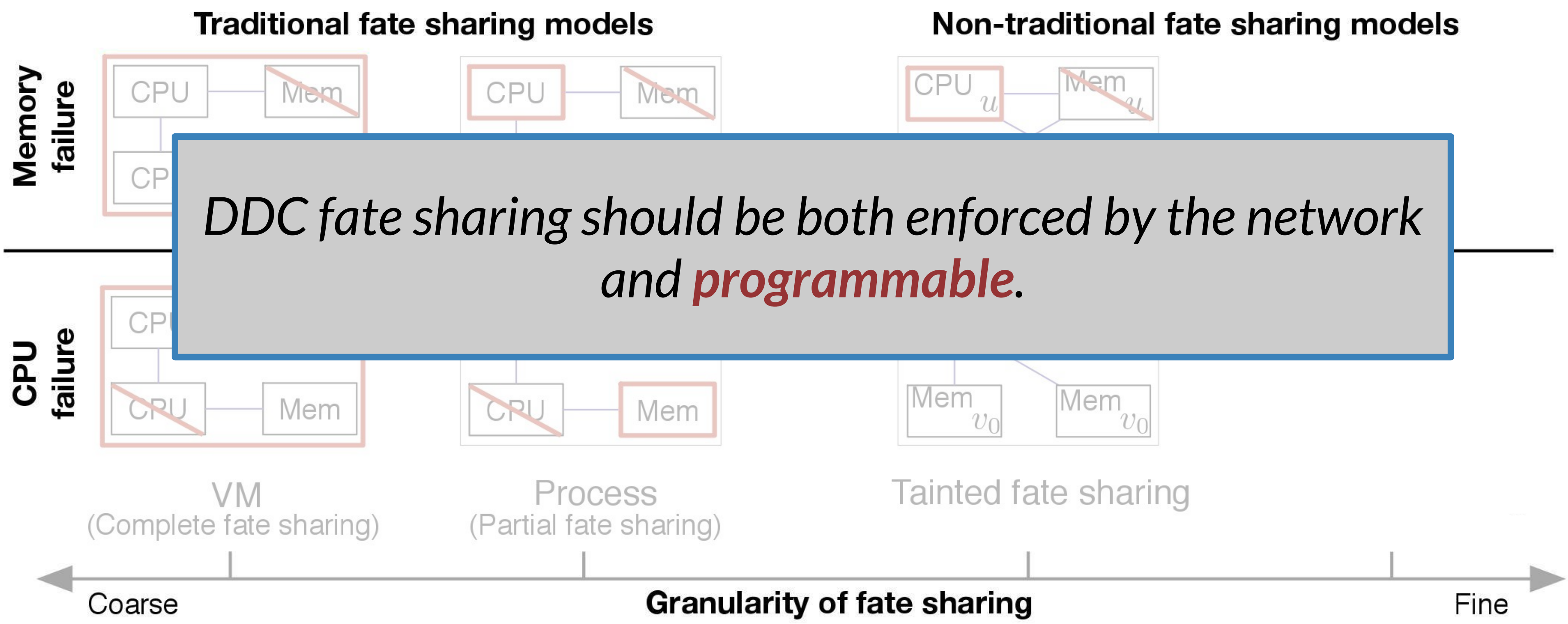


Memory failure



CPU failure

Fate Sharing Granularities

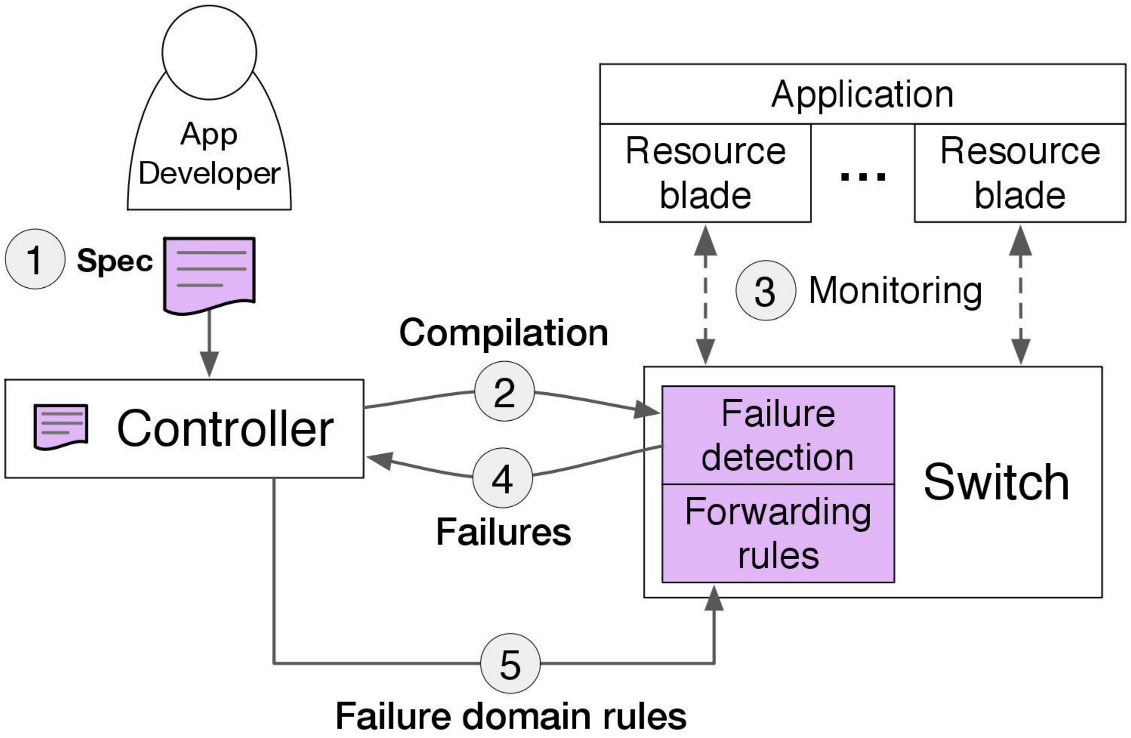


DDC fate sharing should be both enforced by the network and ***programmable****.*

Programmable Fate Sharing

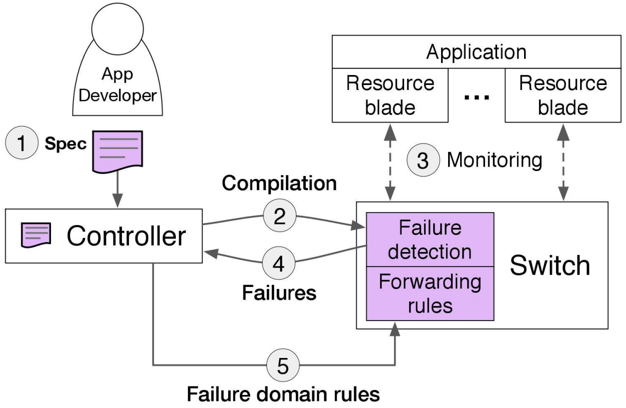
- **Goal:** can describe an arbitrary fate sharing model and install in the network
- Model specification includes
  - Failure detection
  - Failure domain
  - Failure mitigation (optional)
- **Open research questions:**
  - Who should define the specification?
  - What workflow should be used for transformation of specification to switch machine code?

Proposed Workflow



Fate Sharing Specification

- Provides interface between components
- High-level language → high-level networking language [1] → compiles to switch
- **Open research questions:**
  - Spec verification?
  - Language and switch requirements for expressiveness?

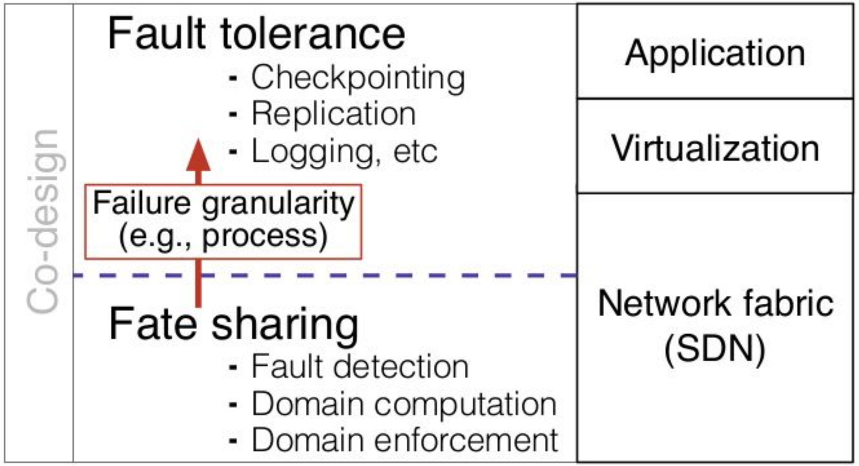


[1] FatTire HotSDN'13, NetKAT POPL'14, Merlin CoNEXT'14, P4 CCR'14, SNAP SIGCOMM'16

Vision: *programmable*, *in-network* fate sharing

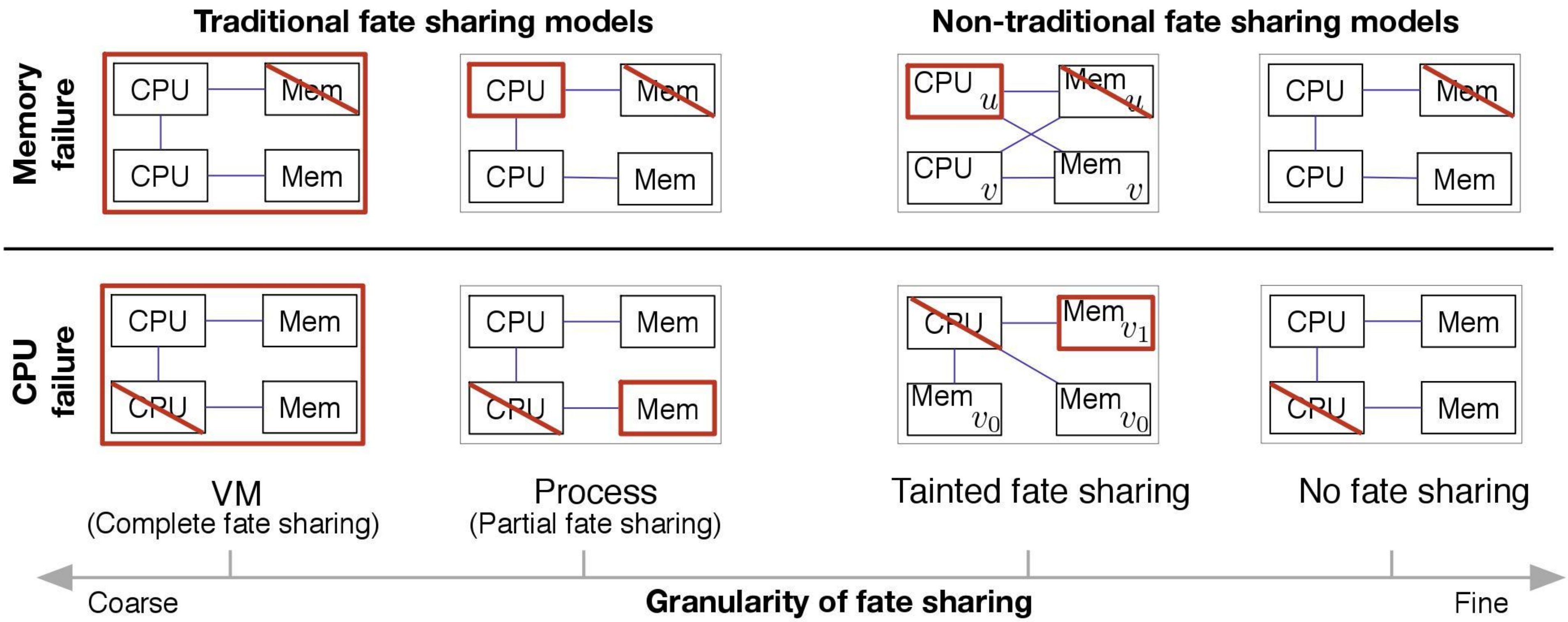
Open research questions

- ▷ Failure semantics for GPUs? Storage?
- ▷ Switch or controller failure?
- ▷ Correlated failures?
- ▷ Other non-traditional fate sharing models?



Thank you!

Backup slides


Traditional fate sharing models
Non-traditional fate sharing models
Memory failure
CPU failure
CPU
Mem
u
v
v1
v0
VM
(Complete fate sharing)
Process
(Partial fate sharing)
Tainted fate sharing
No fate sharing
Coarse
Granularity of fate sharing
Fine

In-Network Memory Replication

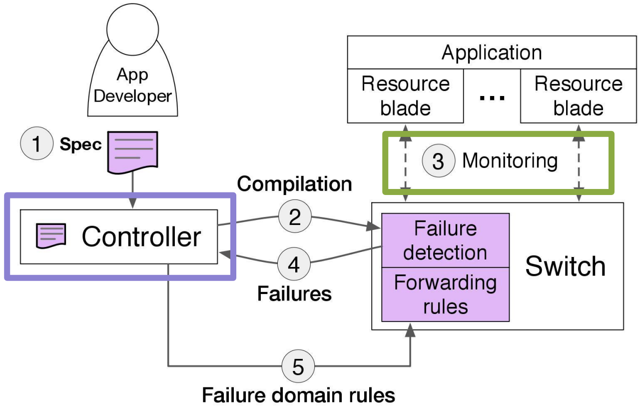
- Port mirror CPU operations to memory replicas, automatically recovers replica during failure
- **Challenges:** coherency, network delay, etc.
- Different assumptions than previous work
  - Persistent storage backings [Sinfonia SOSP'07, RAMCloud SOSP'11, FaRM NSDI'14, Infiniswap NSDI'17]
- Must consider network requirements
  - Combined solutions [GFS OSDI'03, Ceph OSDI'06]
  - Performance sensitive [Costa et. al. OSDI'96]

In-Network CPU Checkpointing

- Controller checkpoints processor state to remote memory (state attached operation packets)
- **Challenges:** consistent client view, checkpoint retention, non-idempotent operations, etc.
- Different requirements than previous work
  - Low tail-latency [Remus NSDI’08, Bressoud et. al. SOSP’95]
- Similar trade-offs (application specific vs generality)
  - Protocol [DMTCP IPDPS’09, Bronevetskey et. al. PPoPP’03]
  - Workflow [Shen et. al. VLDB’14, Xu et. al. ICDE’16]

Passive Application Monitoring

- Defines what information must be collected during normal execution
  - Domain table
  - Context information
  - Application protocol headers

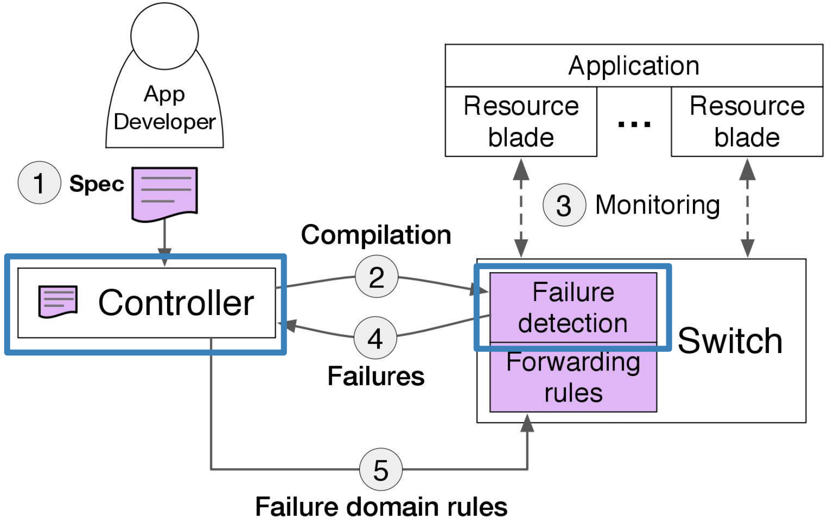


cpu_ip	memory_ip	start	ack
x.x.x.x	x.x.x.x	t_s	t_a

src IP	src port	dst IP	dst port	rtype	op	tstamp

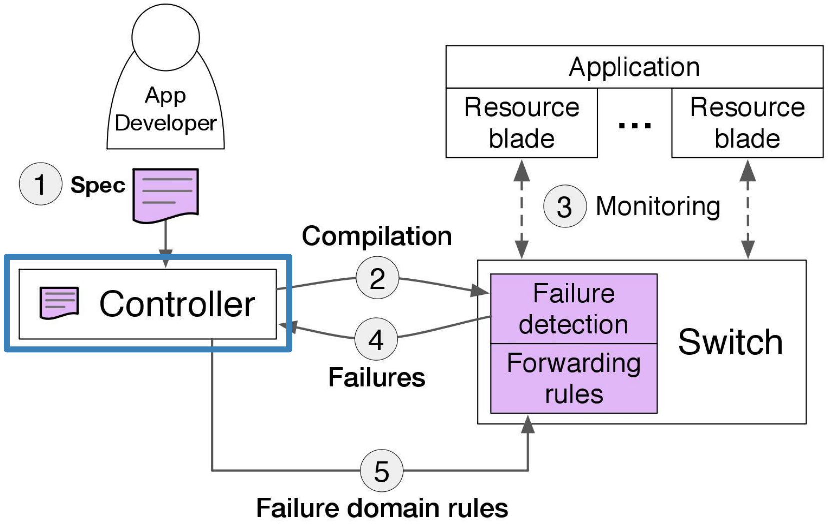
Application Failure Notification

- Spec defines notification semantics
- When controller gets notified of failure → notifies application



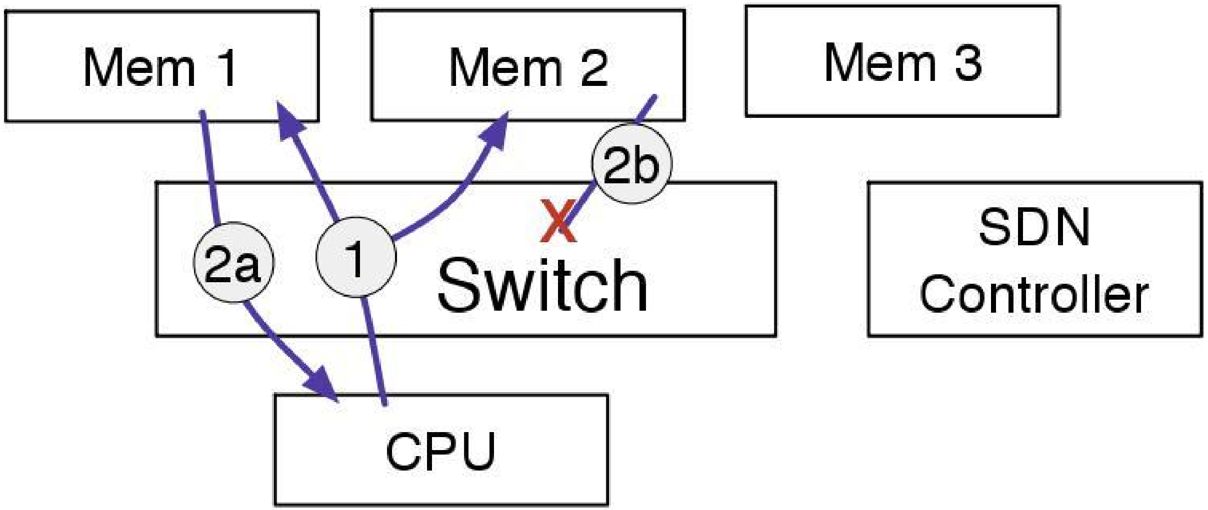
Active Failure Mitigation

- Defines how to generate a failure domain and what rules to install on the switch
- Compares every `domain` entry to failed resource to build failure domain
- Installs rules based on mitigation action



In-Network Memory Recovery

Normal Execution



In-Network Memory Recovery

Under Failure