Labeled RISC-V: A New Perspective on Software-Defined Architecture

Zihao Yu, Bowen Huang, Jiuyue Ma,
Ninghui Sun, Yungang Bao

Oct 14th , 2017 @ Boston

**Institute of Computing Technology (ICT),
Chinese Academy of Sciences (CAS)**

Control in Computer Architecture

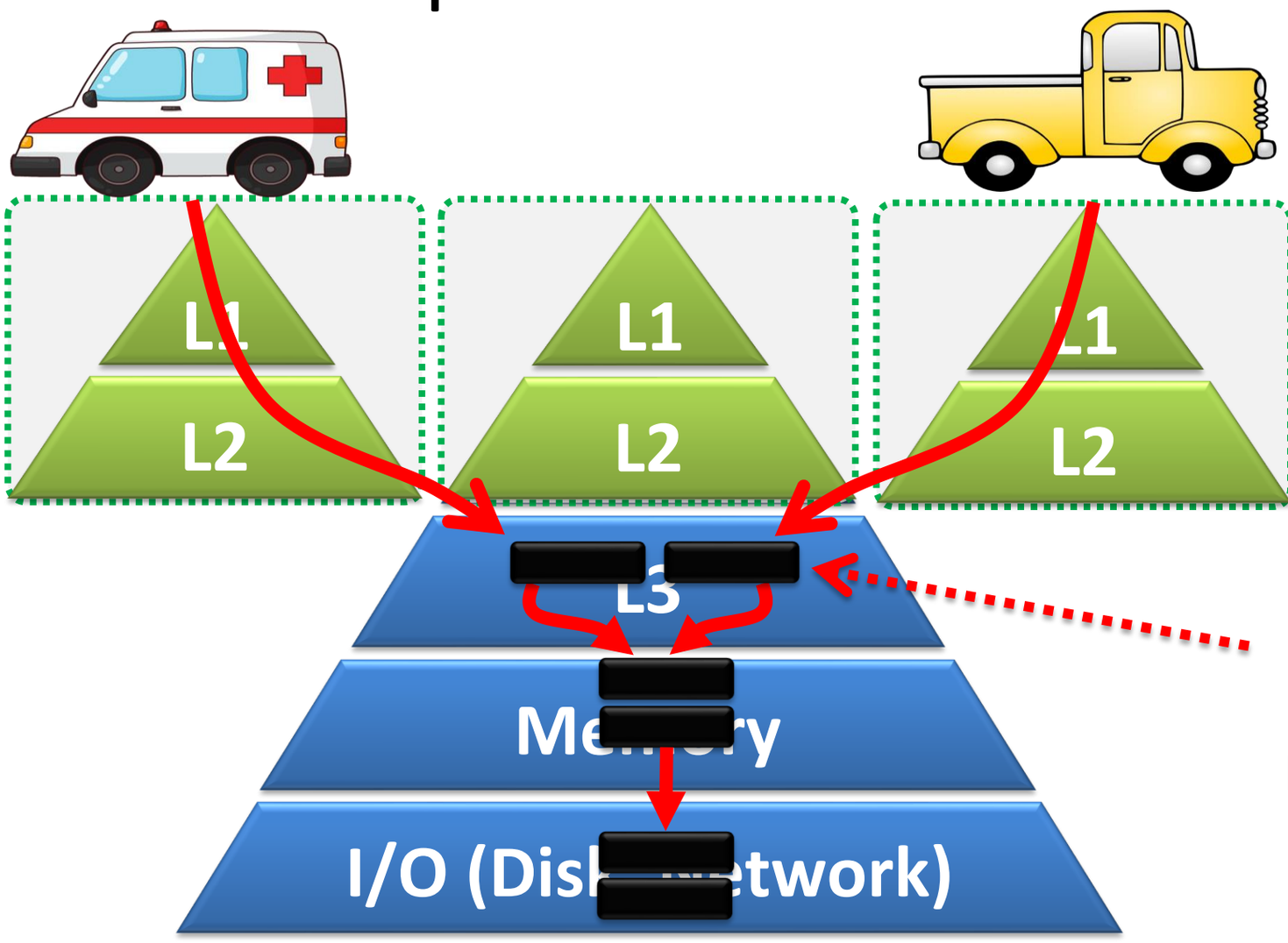
- Maurice Wilkes proposed microprogramming to design the control unit of a processor in 1951.

- RISC became popular in 1980s.
  - More efforts to datapath
  - Less attentions to control

- But as multicore emerges, weak control leads to a new problem.

Weak Control -> Unmanaged Sharing

- Unmanaged sharing -> resource contentions
  -> performance interference/degradation
  -> bad user experience



Which one is more critical?

Unmanaged Sharing in Datacenter

- SMT, LLC, DRAM, Network

websearch

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	134%	103%	96%	96%	109%	102%	100%	96%	96%	104%	99%	100%	101%	100%	104%	103%	104%	103%	99%
LLC (med)	152%	106%	99%	99%	116%	111%	109%	103%	105%	116%	109%	108%	107%	110%	123%	125%	114%	111%	101%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	264%	222%	123%	102%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	270%	228%	122%	103%
HyperThread	81%	109%	106%	106%	104%	113%	106%	114%	113%	105%	114%	117%	118%	119%	122%	136%	>300%	>300%	>300%
CPU power	190%	124%	110%	107%	134%	115%	106%	108%	102%	114%	107%	105%	104%	101%	105%	100%	98%	99%	97%
Network	35%	35%	36%	36%	36%	36%	36%	37%	37%	38%	39%	41%	44%	48%	51%	55%	58%	64%	95%
brain	158%	165%	157%	173%	160%	168%	180%	230%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%

ml_cluster

	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	101%	88%	99%	84%	91%	110%	96%	93%	100%	216%	117%	106%	119%	105%	182%	206%	109%	202%	203%
LLC (med)	98%	88%	102%	91%	112%	115%	105%	104%	111%	>300%	282%	212%	237%	220%	220%	212%	215%	205%	201%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	276%	250%	223%	214%	206%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	287%	230%	223%	211%
HyperThread	113%	109%	110%	111%	104%	100%	97%	107%	111%	112%	114%	114%	114%	119%	121%	130%	259%	262%	262%
CPU power	112%	101%	97%	89%	91%	86%	89%	90%	89%	92%	91%	90%	89%	89%	90%	92%	94%	97%	106%
Network	57%	56%	58%	60%	58%	58%	58%	58%	59%	59%	59%	59%	59%	63%	63%	67%	76%	89%	113%
brain	151%	149%	174%	189%	193%	202%	209%	217%	225%	239%	>300%	>300%	279%	>300%	>300%	>300%	>300%	>300%	>300%

memkeyval

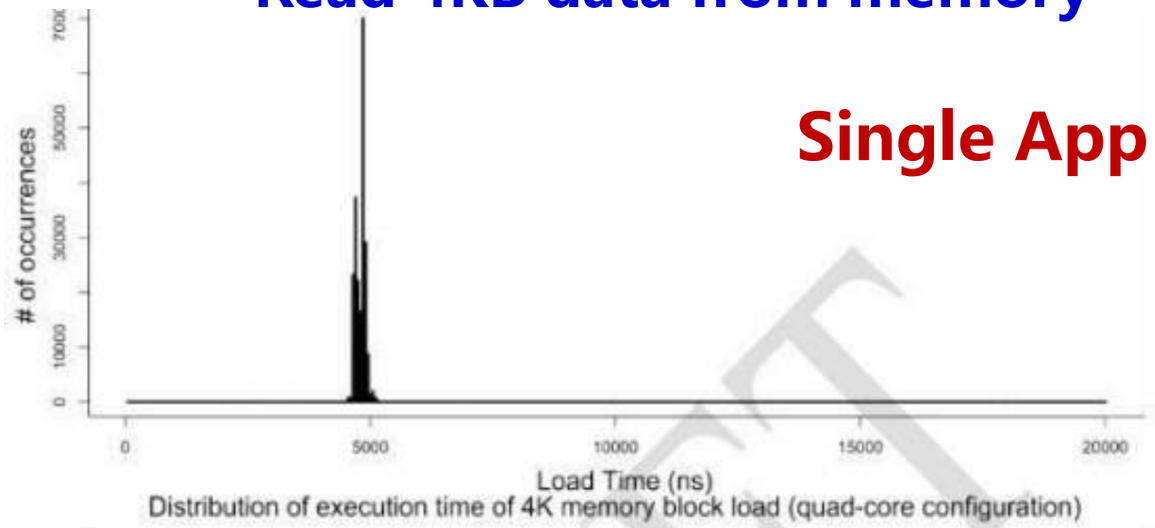
	5%	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%
LLC (small)	115%	88%	88%	91%	99%	101%	79%	91%	97%	101%	135%	138%	148%	140%	134%	150%	114%	78%	70%
LLC (med)	209%	148%	159%	107%	207%	119%	96%	108%	117%	138%	170%	230%	182%	181%	167%	162%	144%	100%	104%
LLC (big)	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	280%	225%	222%	170%	79%	85%
DRAM	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	252%	234%	199%	103%	100%
HyperThread	26%	31%	32%	32%	32%	32%	33%	35%	39%	43%	48%	51%	56%	62%	81%	119%	116%	153%	>300%
CPU power	192%	277%	237%	294%	>300%	>300%	219%	>300%	292%	224%	>300%	252%	227%	193%	163%	167%	122%	82%	123%
Network	27%	28%	28%	29%	29%	27%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%
brain	197%	232%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%	>300%

Lo et al. Heracles: Improving Resource Efficiency at Scale, ISCA, 2015.

Impact on real time

- Hard to satisfy with multicore
  - Disable multicore

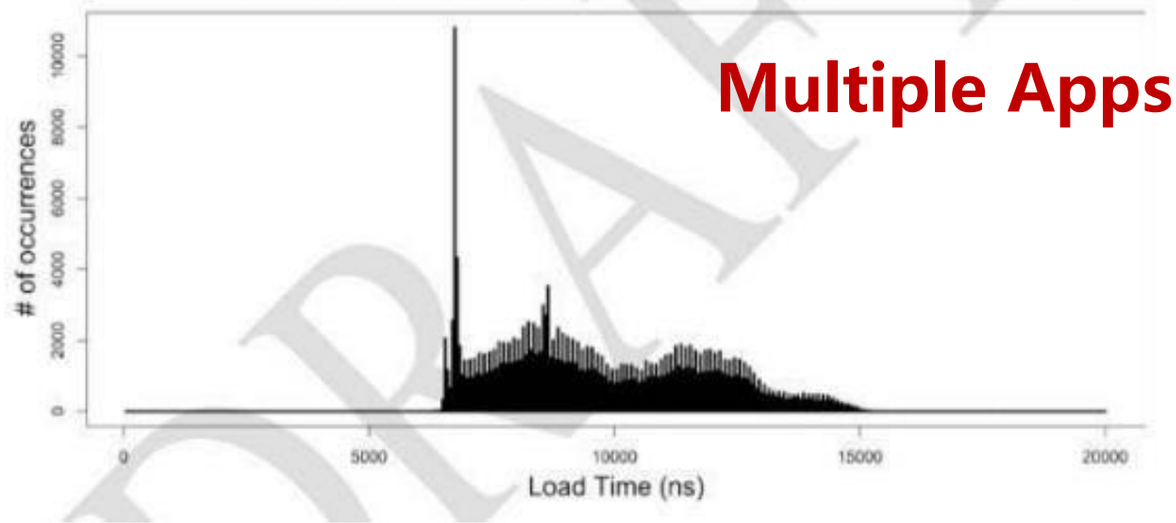
Read 4KB data from memory

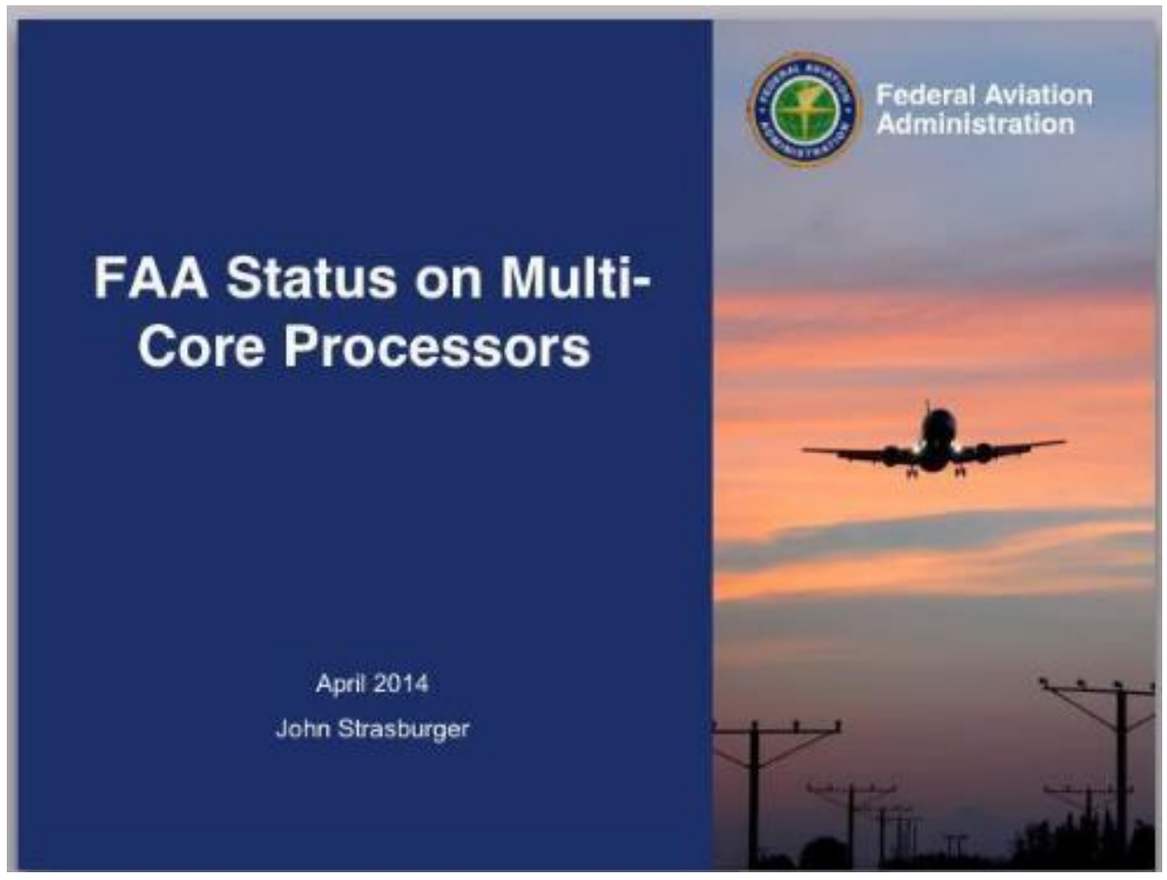


Distribution of execution time of 4K memory block load (quad-core configuration)

Single App

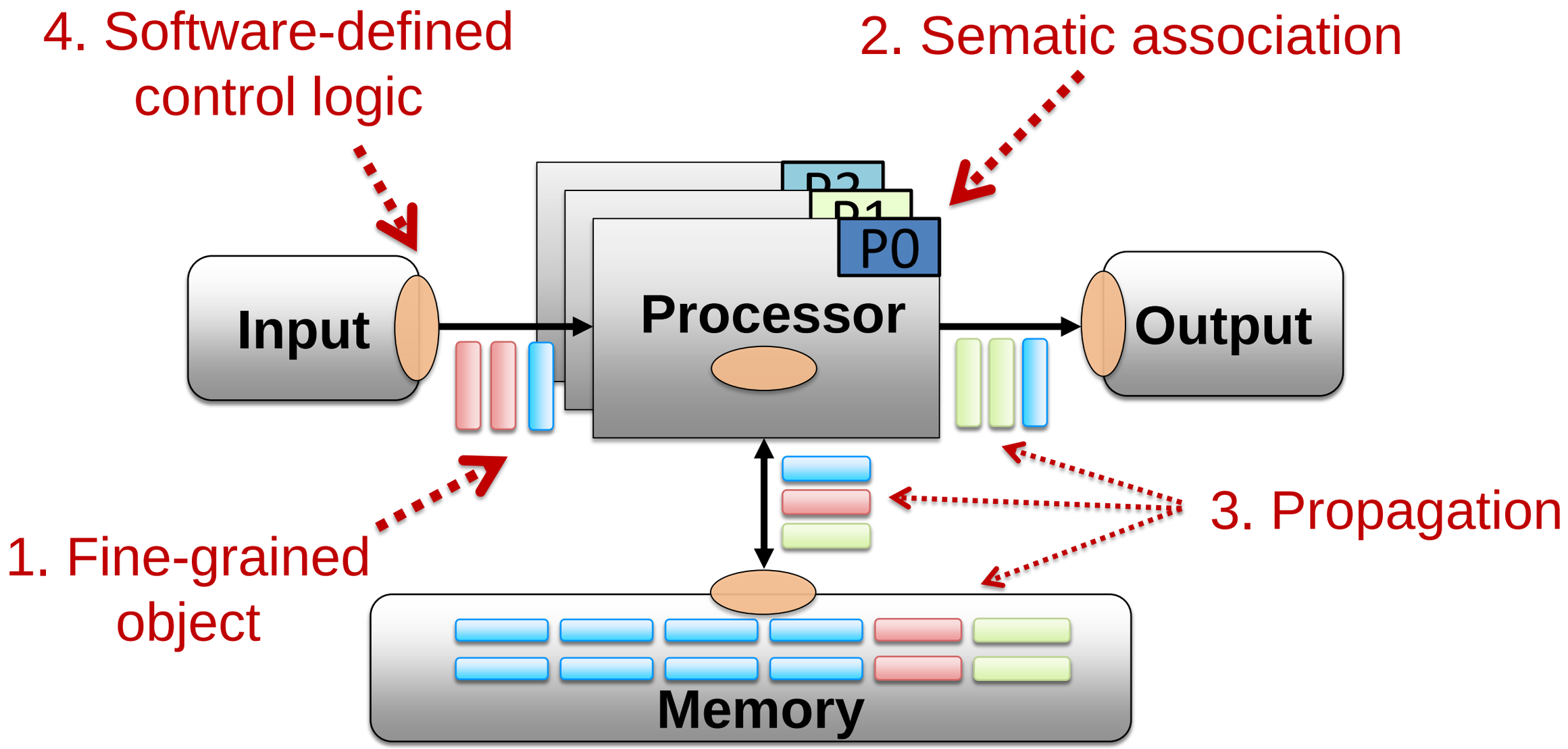
Multiple Apps





White Paper on Issues Associated with Interference Applied to Multicore Processors, 2014

Labeled von Neumann Architecture (LvNA)



Bao and Wang, Labeled von Neumann Architecture for Software-Defined Cloud, Journal of Computer Science and Technology, 2017 Vol. 32 (2): 219-223.

Programmable Architecture for Resourcing-on-Demand

PARD

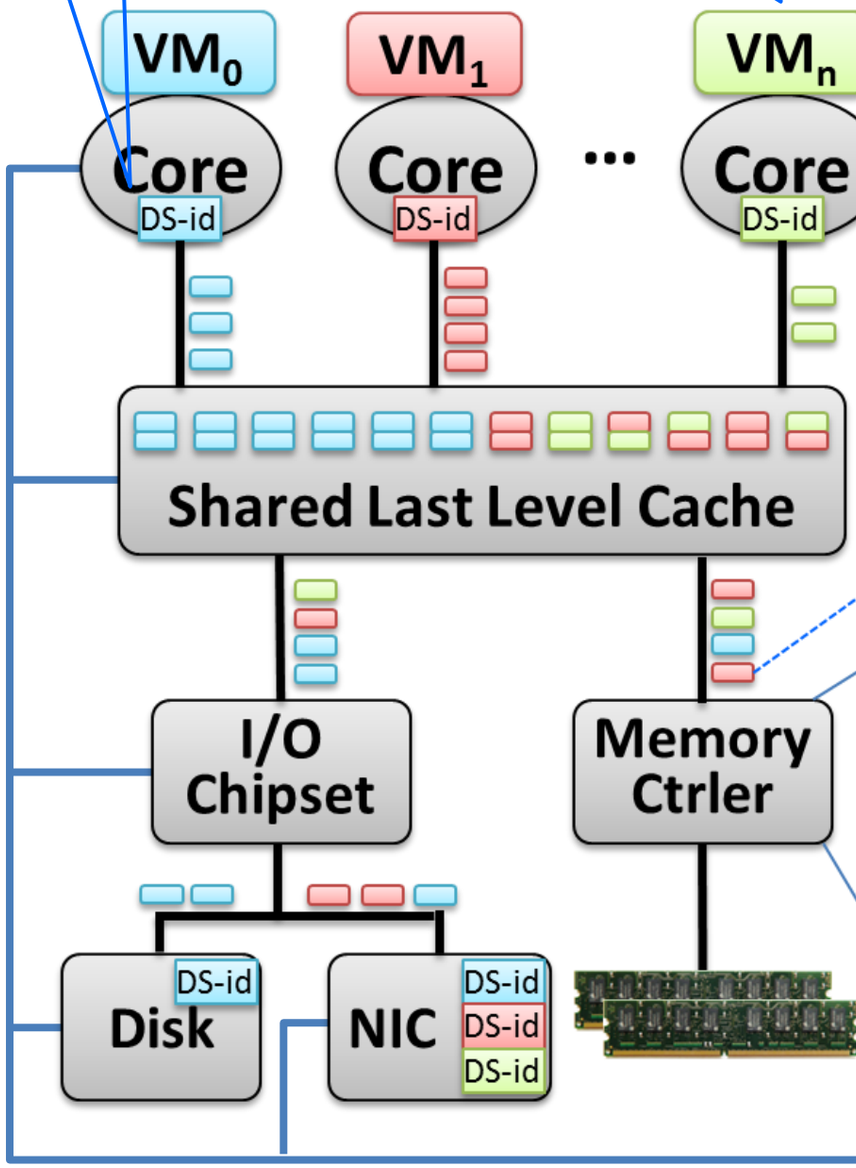
① Add label Reg

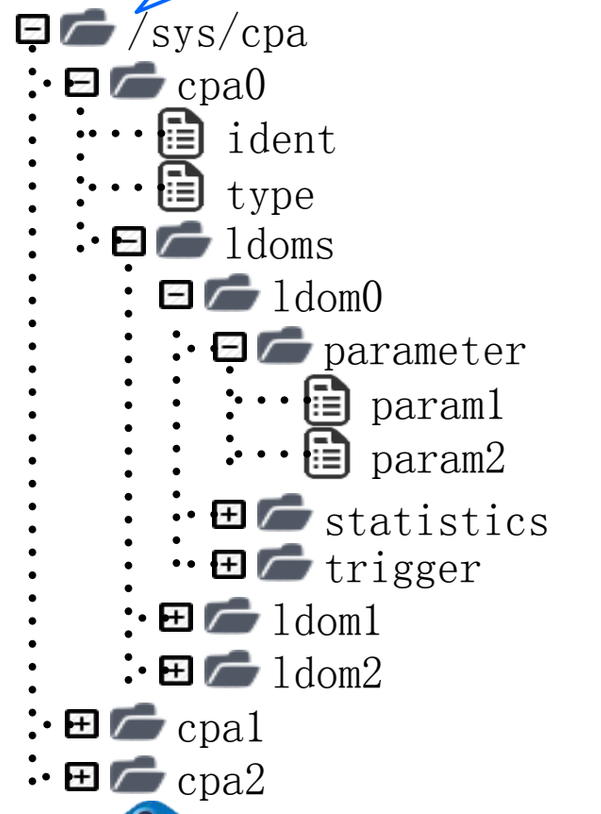
② Allocate label to each VM

③ Attach label to each Req

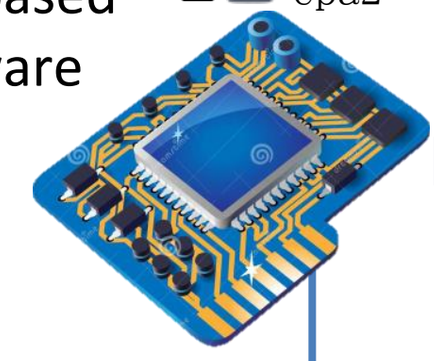
④ Add label-based, programmable control logic

⑤ Abstract label Regs and CLs into files





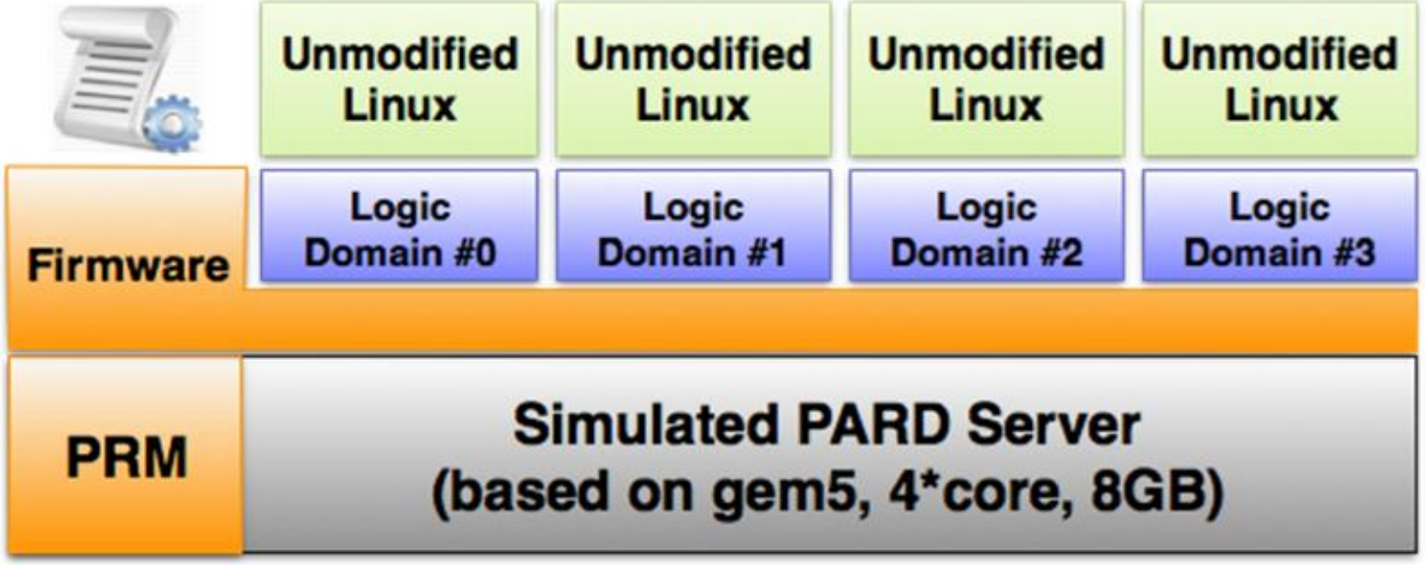
Linux-based firmware

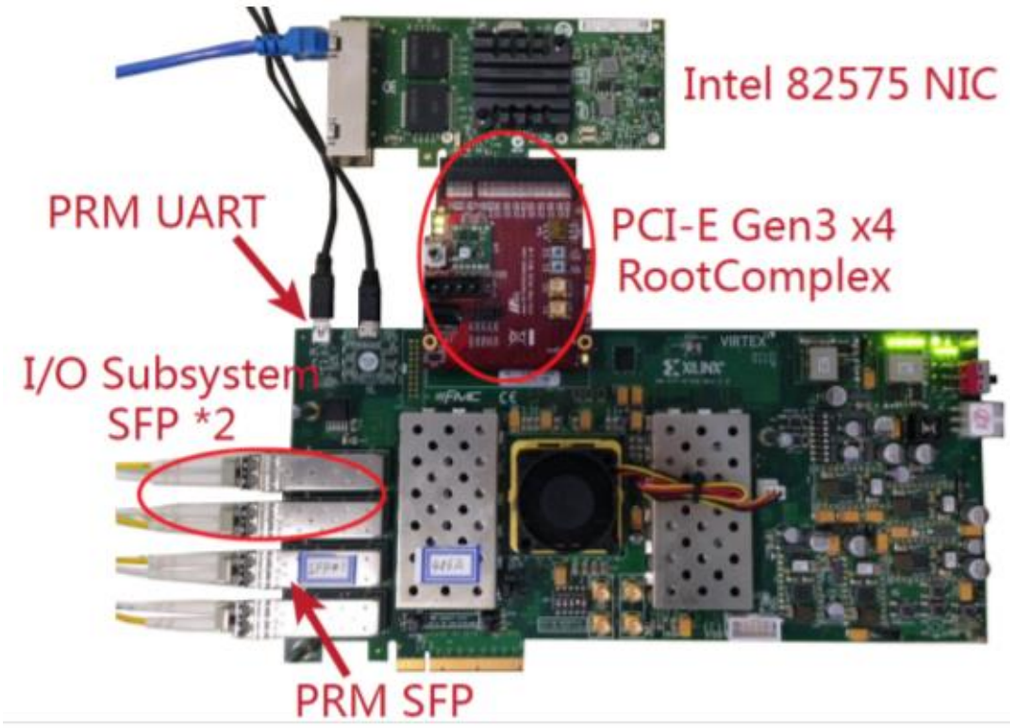


Platform Resource Manager (PRM)

Implementation

- **Full-system cycle-accurate simulator** Open Sourced *
- **FPGA prototype on Xilinx VC709 evaluation board**
  - **MicroBlaze version** Deprecated
  - **RISC-V version** Open Sourced +



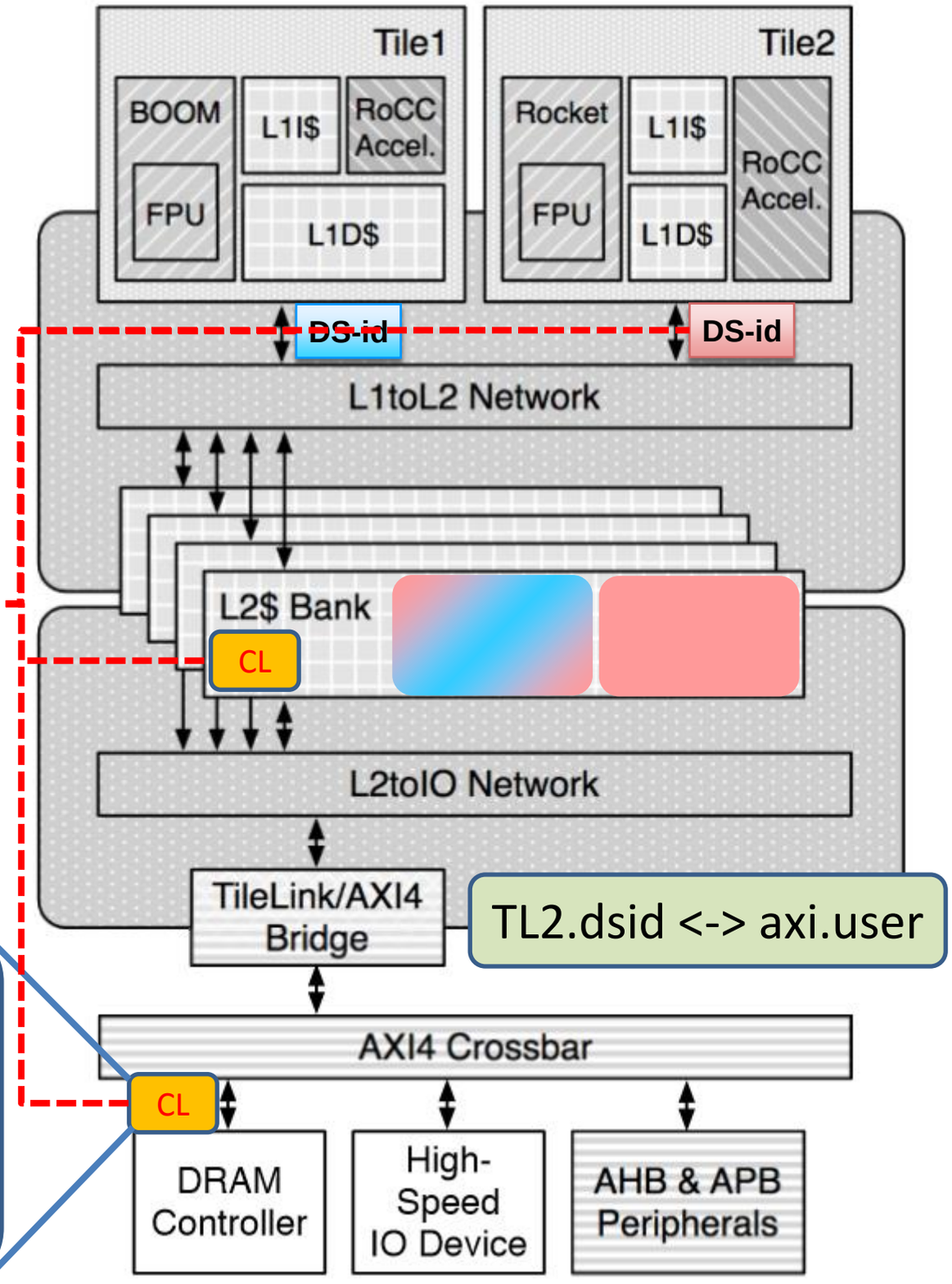


* http://github.com/fsg-ict/PARD-gem5

\+ http://github.com/fsg-ict/labeled-RISC-V

LvNA + RISC-V = Labeled RISC-V

- Features to add
  - Labels registers after tiles
  - Cache CL
  - Label Converter in TLtoAXI
  - Memory CL
  - PRM

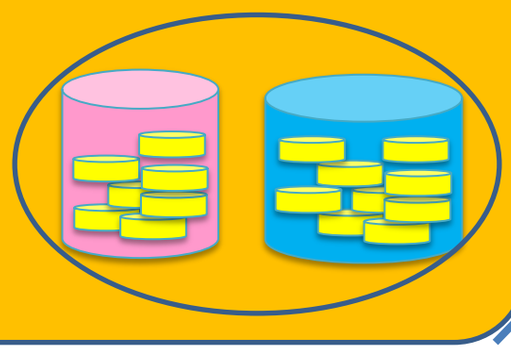


PRM

Address mapping

Labeled token bucket

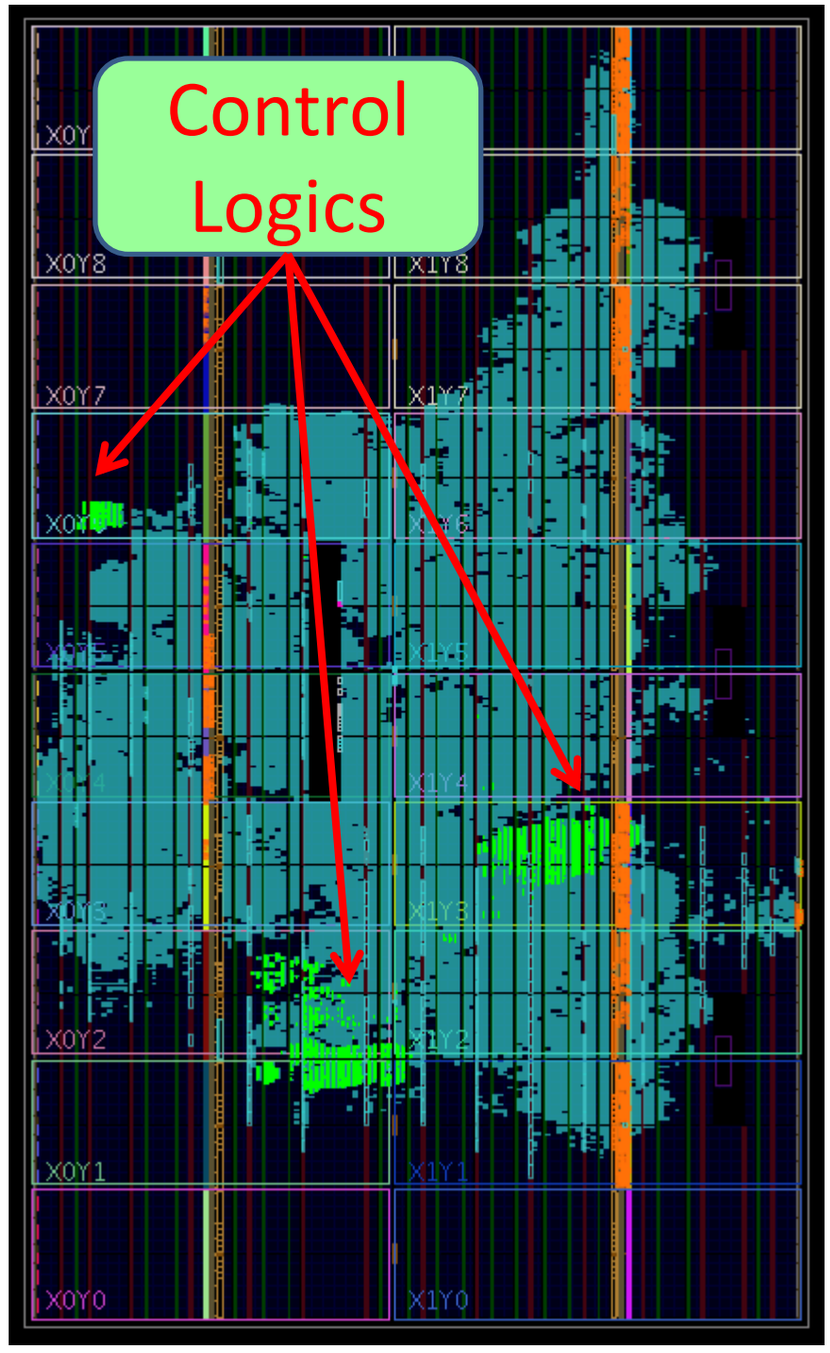
DS-id	Base	Len
1	0x0000	0x4000
2	0x8000	0x8000



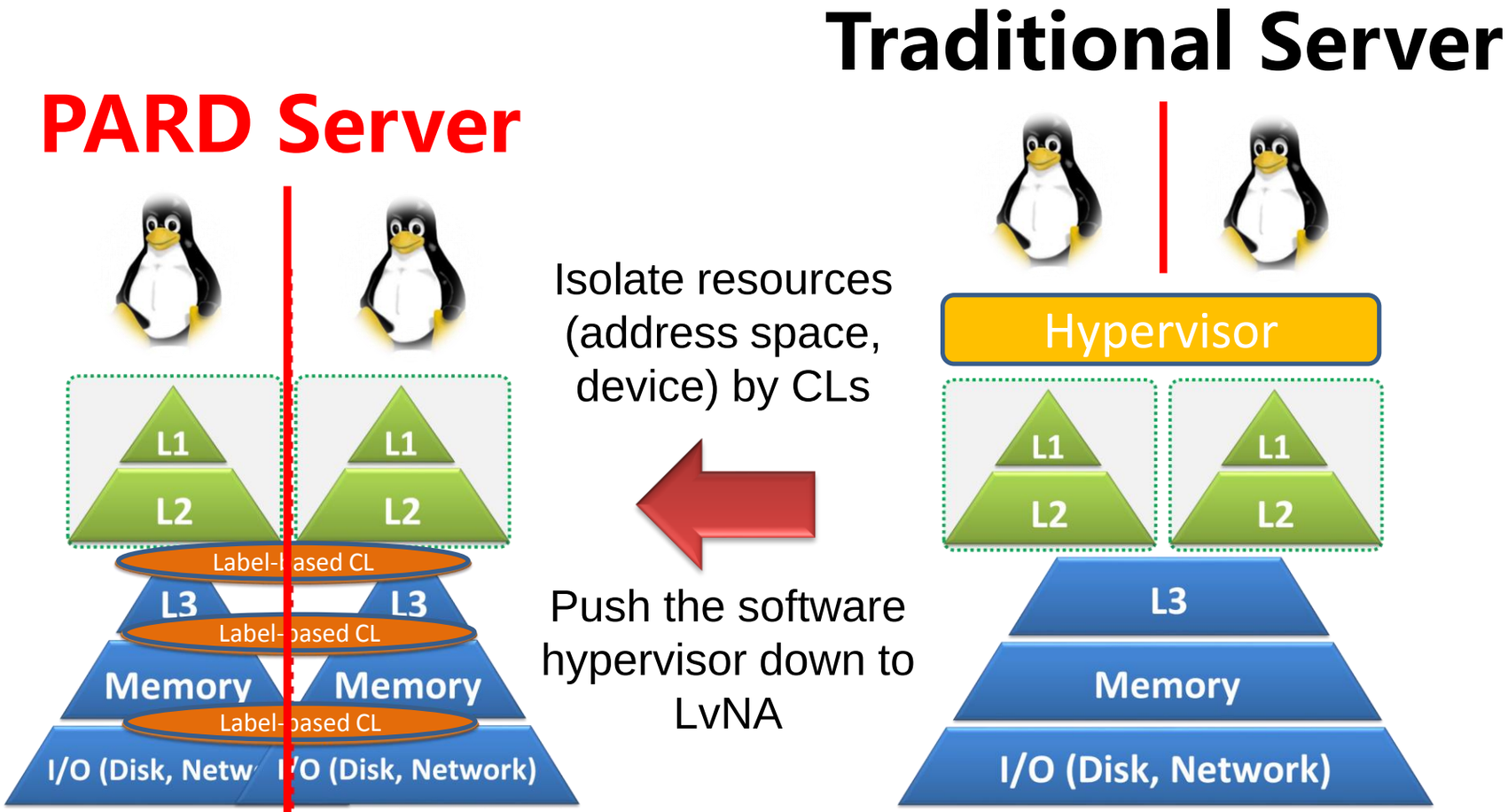
https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf

Overheads

- 16 lines of chisel code to add labels into RocketChip
  - Add dsid member in the Bundle of TileLink2
  - Attach labels at the TileLink2 masters of core tiles
- < 5% resource overheads for CLs
  - Much less with complex cores, e.g. BOOM
- No performance overheads for critical apps according to the timing report



Demo 1 - NoHype




```
root@ldom:~# while true; do dat root@ldom:~#
e; sleep 1; done
```

Demo 2 - Memory Bandwidth Control

- Use labeled token buckets to protect the bandwidth from attacker

Solo

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	53.8	0.074376	0.074305	0.074430
Scale:	46.7	0.086195	0.085691	0.087494
Add:	50.1	0.119921	0.119842	0.120034
Triad:	48.2	0.124578	0.124473	0.124679

interfered

Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	11.7	0.344165	0.340619	0.347746
Scale:	12.1	0.341771	0.331738	0.356621
Add:	8.1	0.748021	0.737555	0.755204
Triad:	8.1	0.751058	0.740938	0.760358

isolated

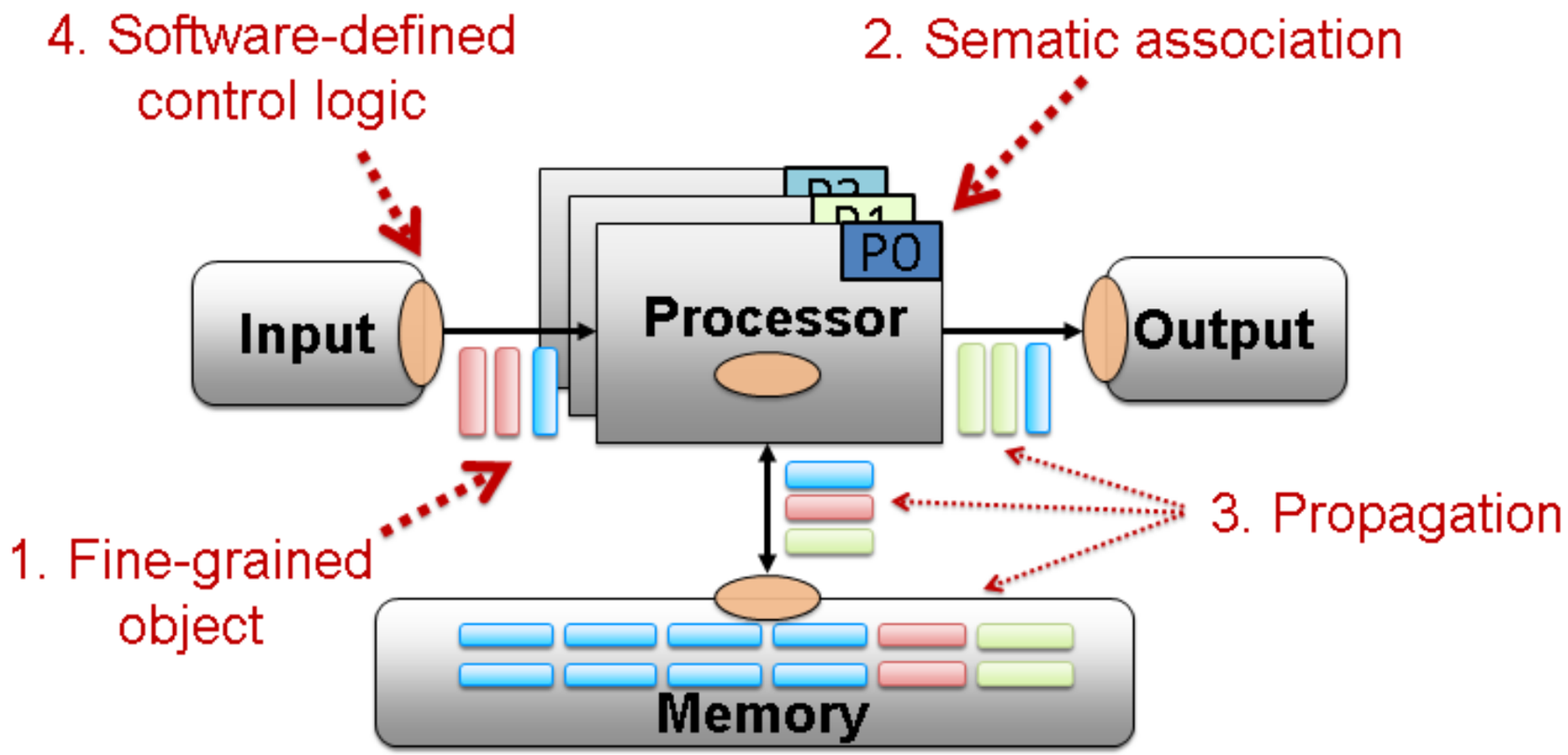
Function	Best Rate MB/s	Avg time	Min time	Max time
Copy:	53.7	0.074583	0.074555	0.074617
Scale:	46.6	0.085922	0.085853	0.086026
Add:	50.0	0.120866	0.119926	0.122406
Triad:	48.1	0.125066	0.124837	0.125717

A lot to explore!

- Finished
- On-going
- Have ideas
- Feature work

- **Theory**: How does LvNA impact on RAM, PRAM, LogP models?
- **Hardware/Arch**: How to implement LvNA at CPU pipeline/SMT, memory, storage, networking? How to correlate LvNA and SDN by labels?
- **OS/Hypervisor**: How to correlate labels with VMs, containers, processes, threads? How to abstract programming interfaces for labels?
- **Programing Model and Compilers**: How to express users' requirements and propagate to the hardware via labels? How to make compilers support labels?
- **Distributed systems**: How to correlate labels with distributed resources? How to manage distributed systems with label mechanisms?
- **Measurement/Audit**: How to leverage labels to gauge and audit resource usages?

Summary



- **LvNA:** a model of software-defined architecture
- **PARD:** a proof of concept of LvNA
- **Labeled RISC-V:** an implementation of LvNA

Thanks

Q & A

yuzihao@ict.ac.cn