EPFL

VLSC VCA

PARENDI

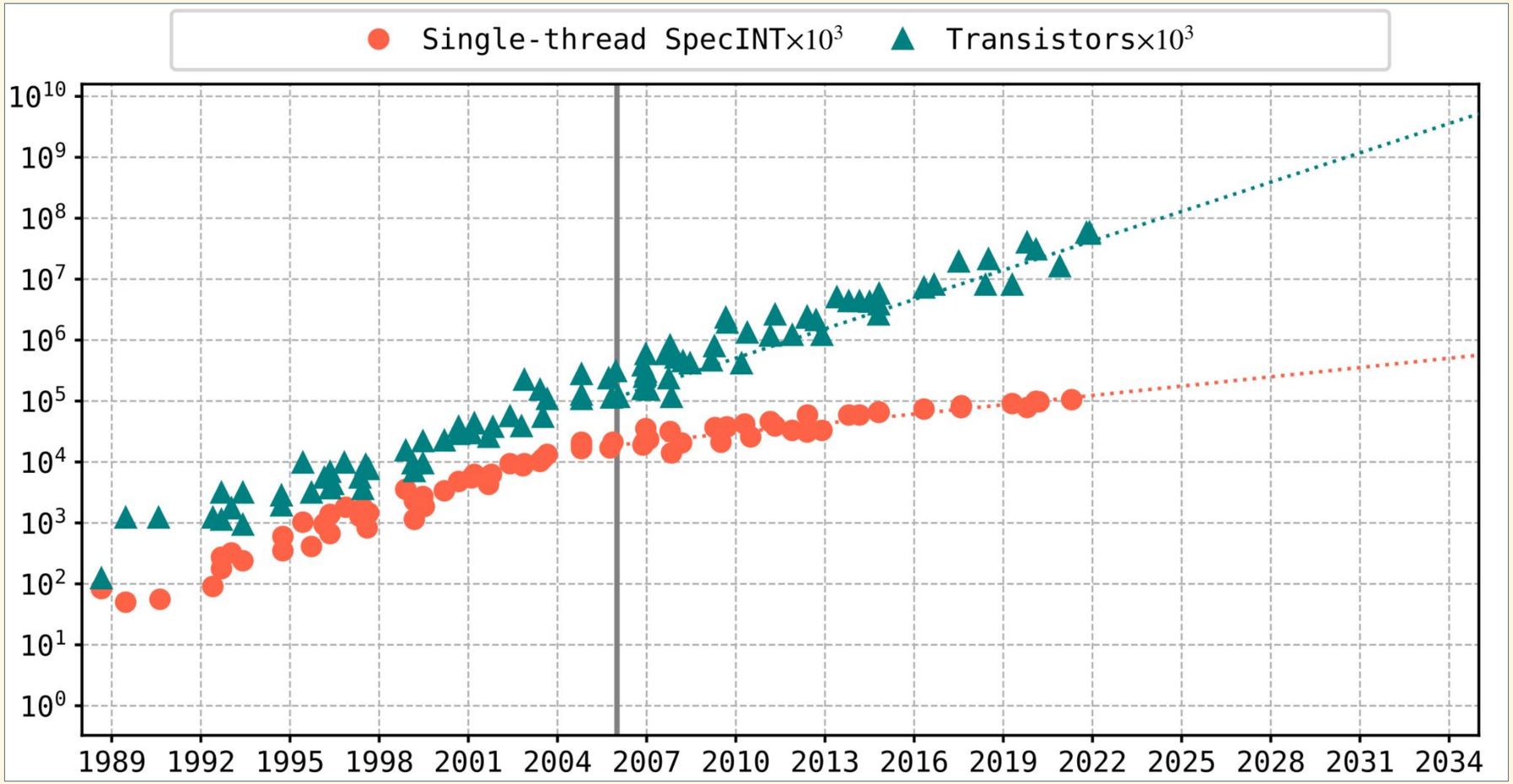
Thousand-Way Parallel RTL Simulation

Mahyar Emami, Thomas Bourgeat, James R. Larus

Chip Simulation is Slow

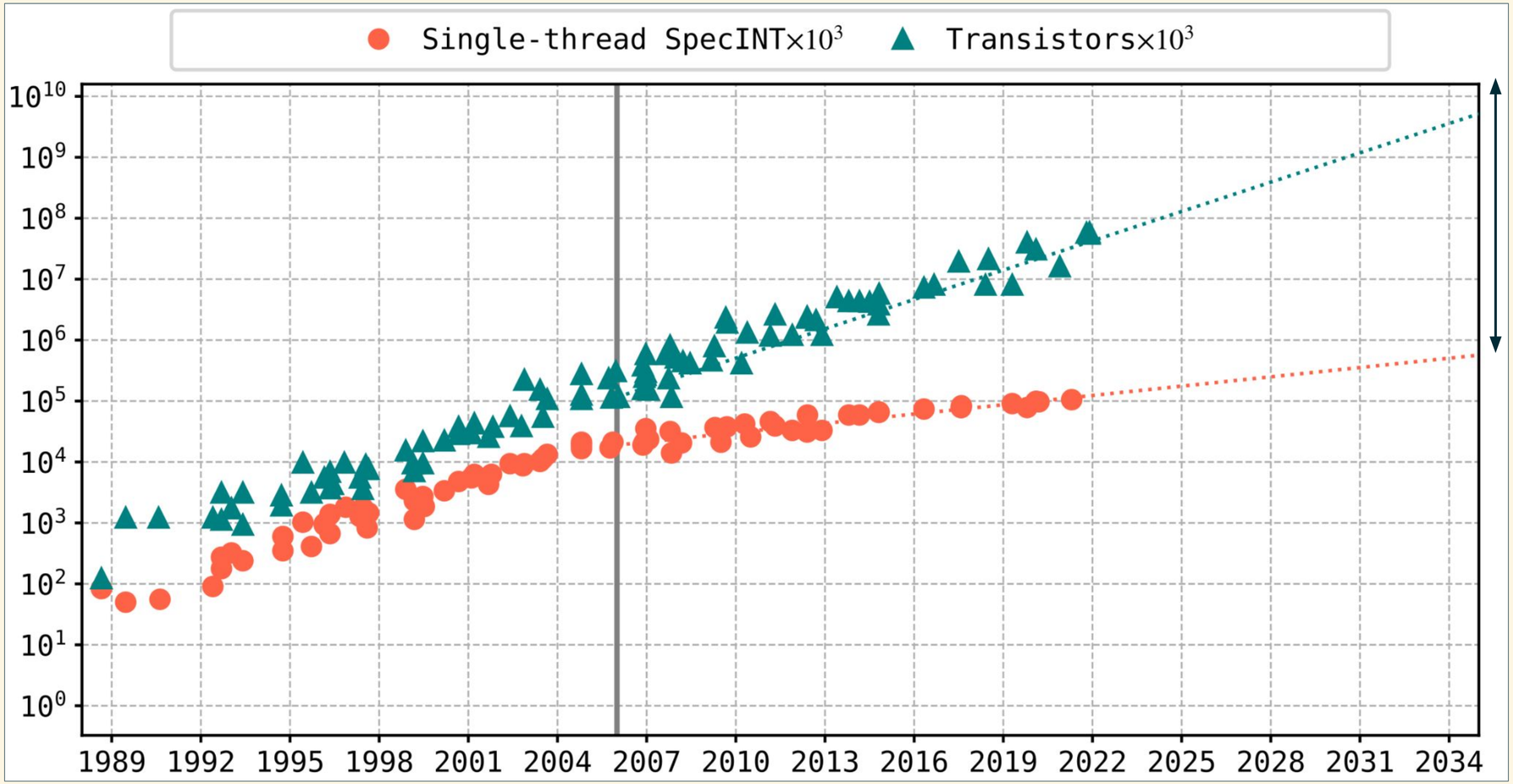
Chip Simulation is ~~Slow~~ Becoming Slower

A Growing Gap



Reproduced from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

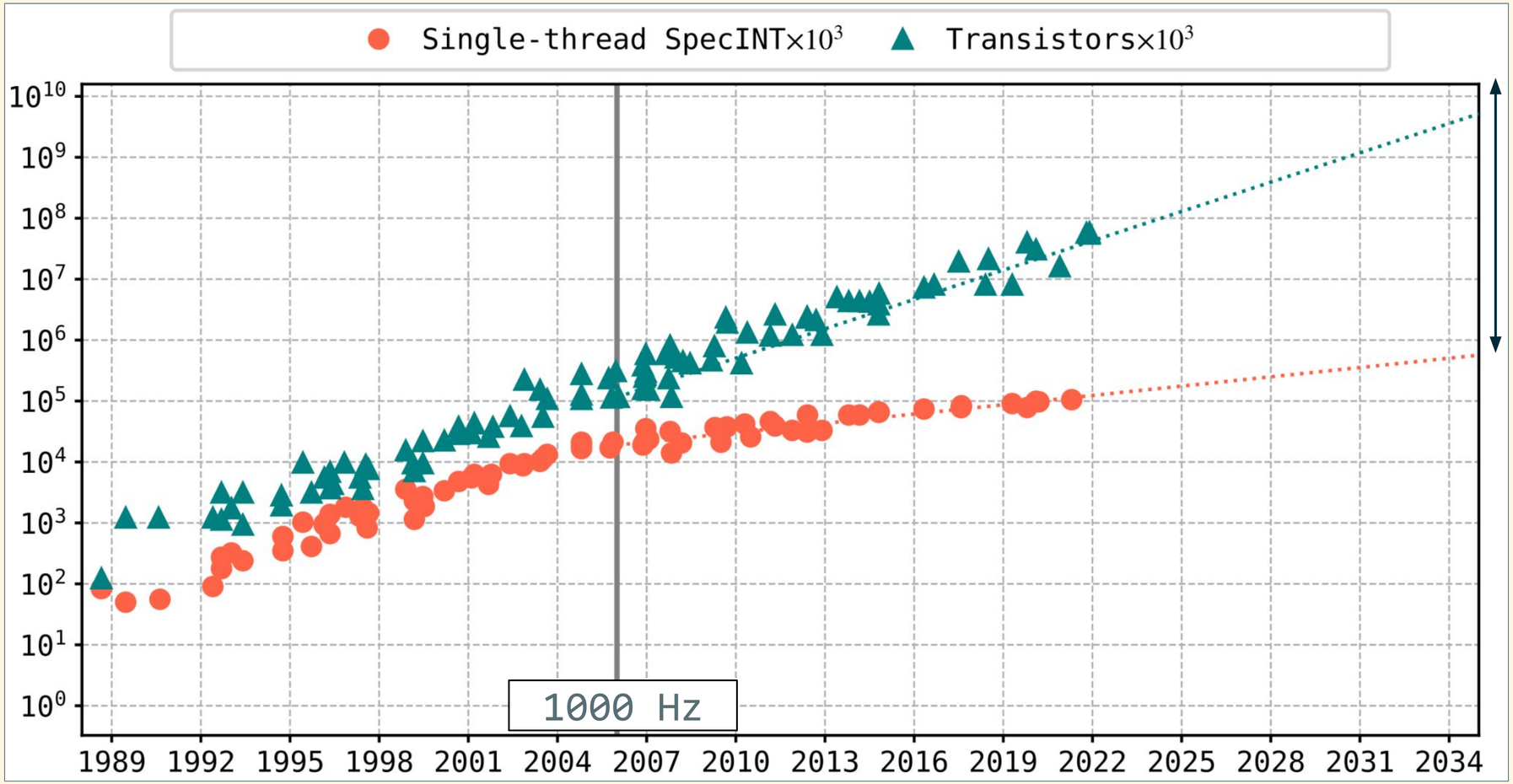
A Growing Gap



≅x1000

Reproduced from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Chip Simulation is Becoming Slower

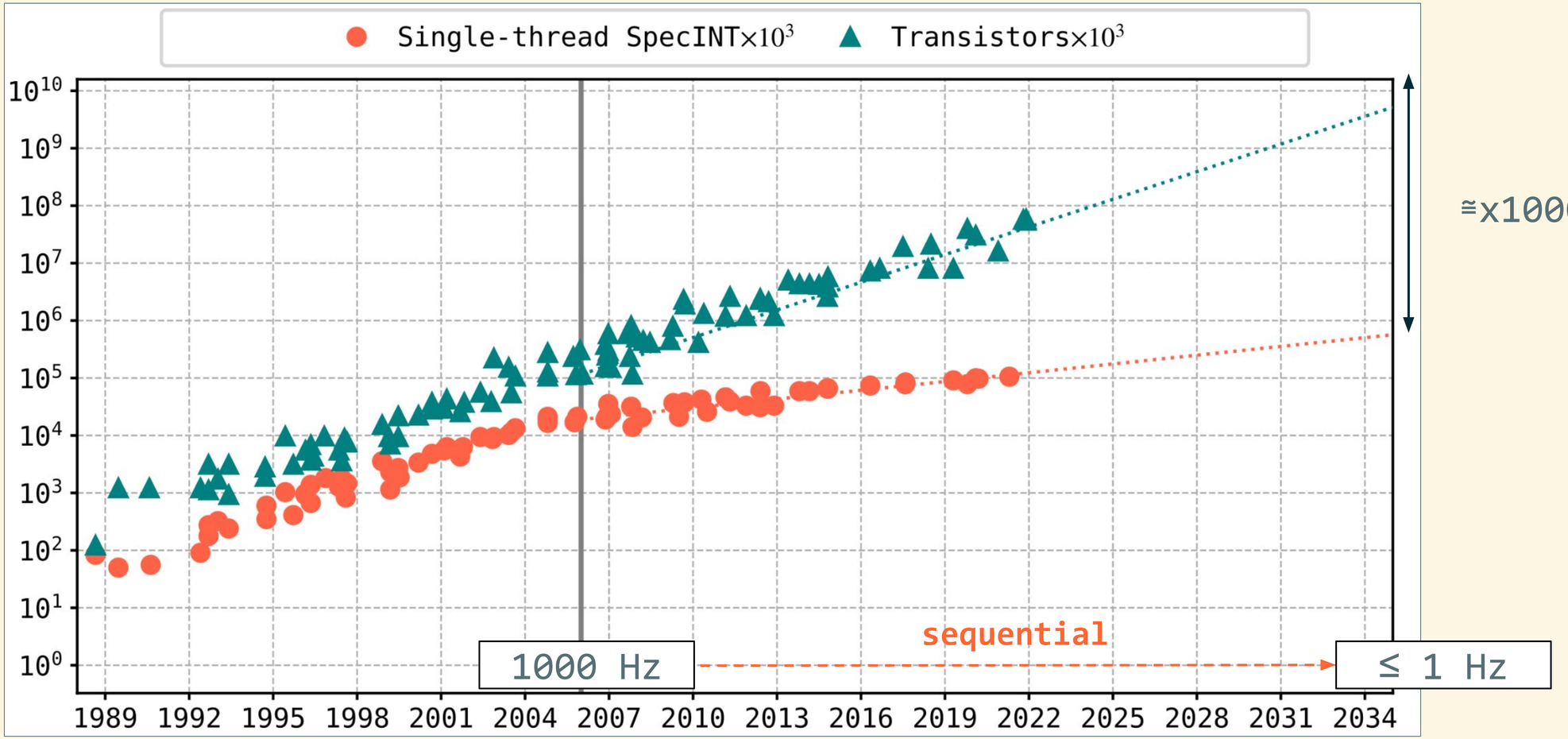


≅x1000

1 GHz chip at 1000 Hz → million times slowdown

Reproduced from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

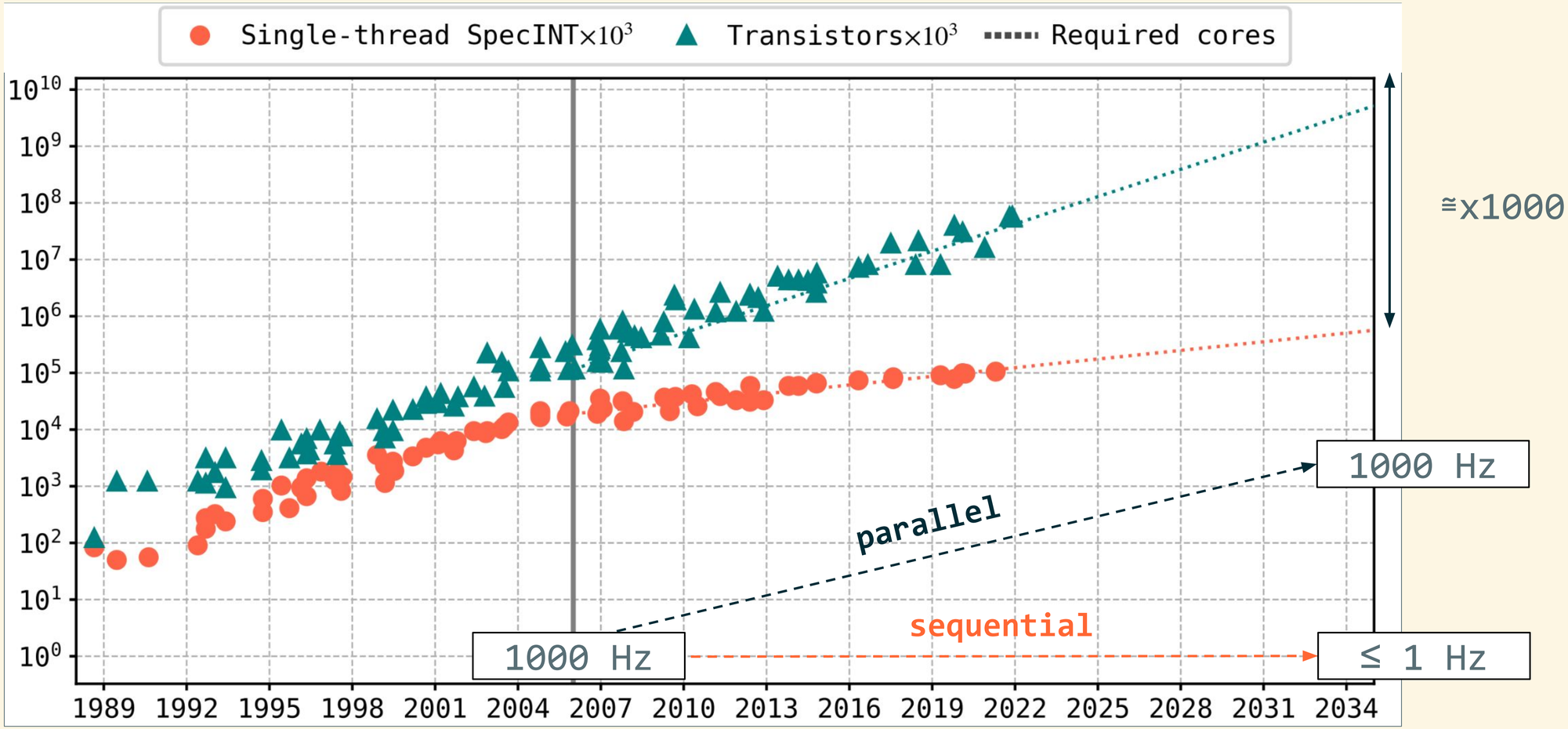
Chip Simulation is Becoming Slower



1 GHz chip at 1000 Hz → million times slowdown

Reproduced from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

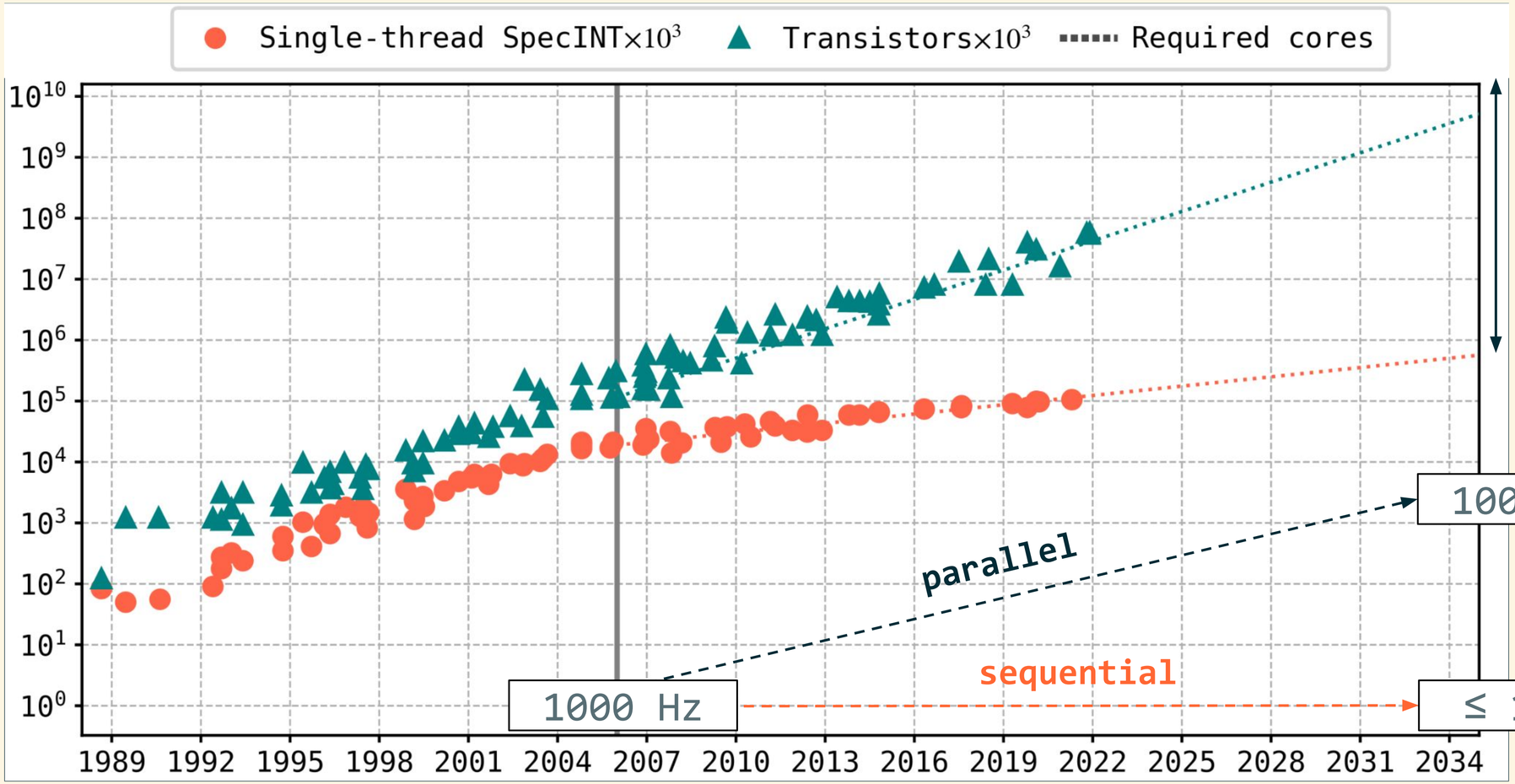
Chip Simulation is Becoming Slower



1 GHz chip at 1000 Hz → million times slowdown

Reproduced from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Chip Simulation is Becoming Slower



≥1000-way parallel

We need 1000-way parallel simulators!

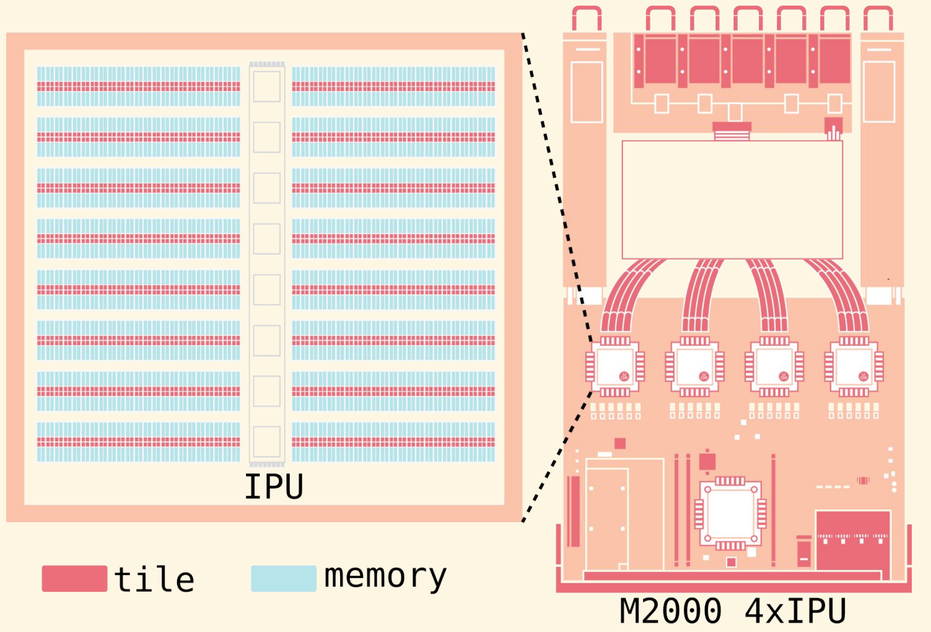
Reproduced from https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/

Parendi: Thousand-Way Parallel RTL Simulation

- Runs RTL simulation on a real thousand-core chip → Graphcore IPU

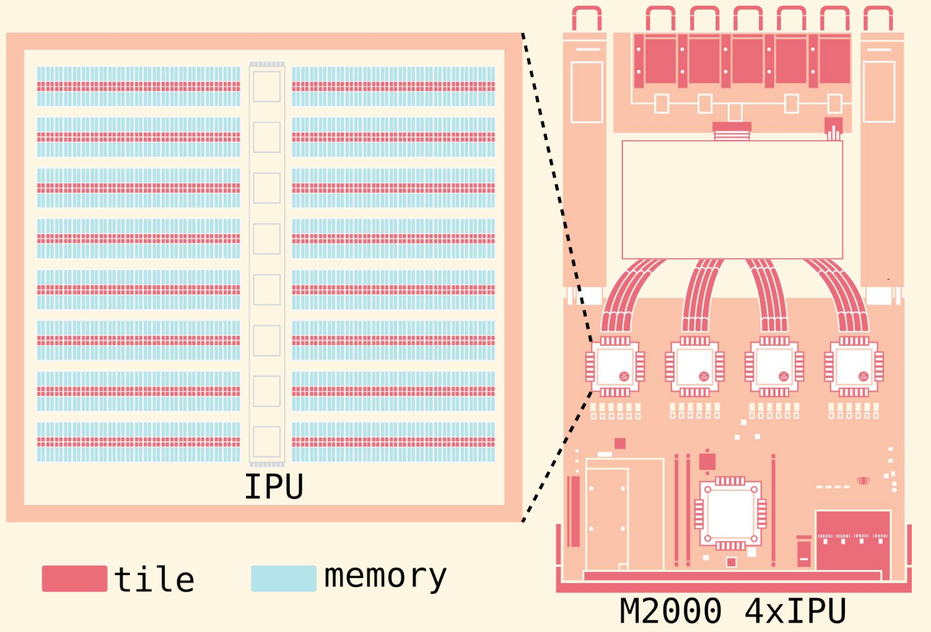
Parendi: Thousand-Way Parallel RTL Simulation

- Runs RTL simulation on a real thousand-core chip → Graphcore IPU
- Graphcore IPU (Intelligence Processing Unit)
  - 1472 cores per socket
  - ≃ 600 KiB of SRAM per core → 900 MiB on-chip
  - Message-passing architecture
    - Bulk-synchronous parallel (BSP) execution model
    - Low-cost synchronization/communication
  - Scale out to **4x (this work)**, 16x, 64x, …



Parendi: Thousand-Way Parallel RTL Simulation

- Runs RTL simulation on a real thousand-core chip → Graphcore IPU
- Graphcore IPU (Intelligence Processing Unit)
  - 1472 cores per socket
  - ≃ 600 KiB of SRAM per core → 900 MiB on-chip
  - Message-passing architecture
    - Bulk-synchronous parallel (BSP) execution model
    - Low-cost synchronization/communication
  - Scale out to **4x (this work)**, 16x, 64x, …
- Parendi
  - Targets the IPU (based on Verilator)
  - 1000-way parallelization and optimizations
  - Up to 4x faster than Verilator on x64

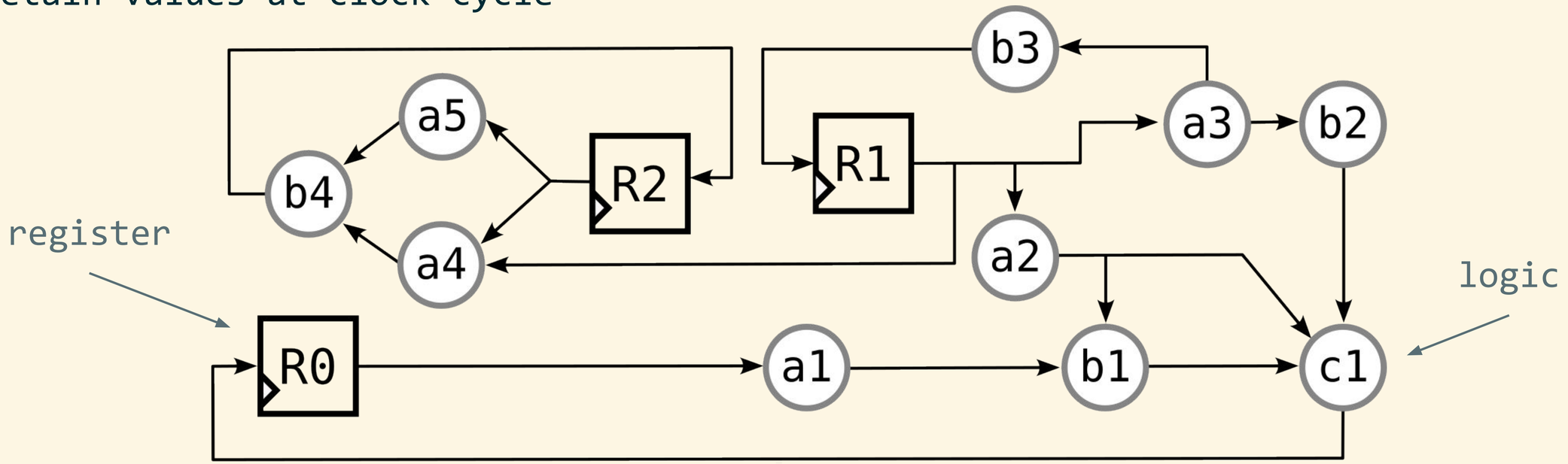


Background

Simulating Chips: What is RTL?

Digital hardware is described in RTL (register transfer level)

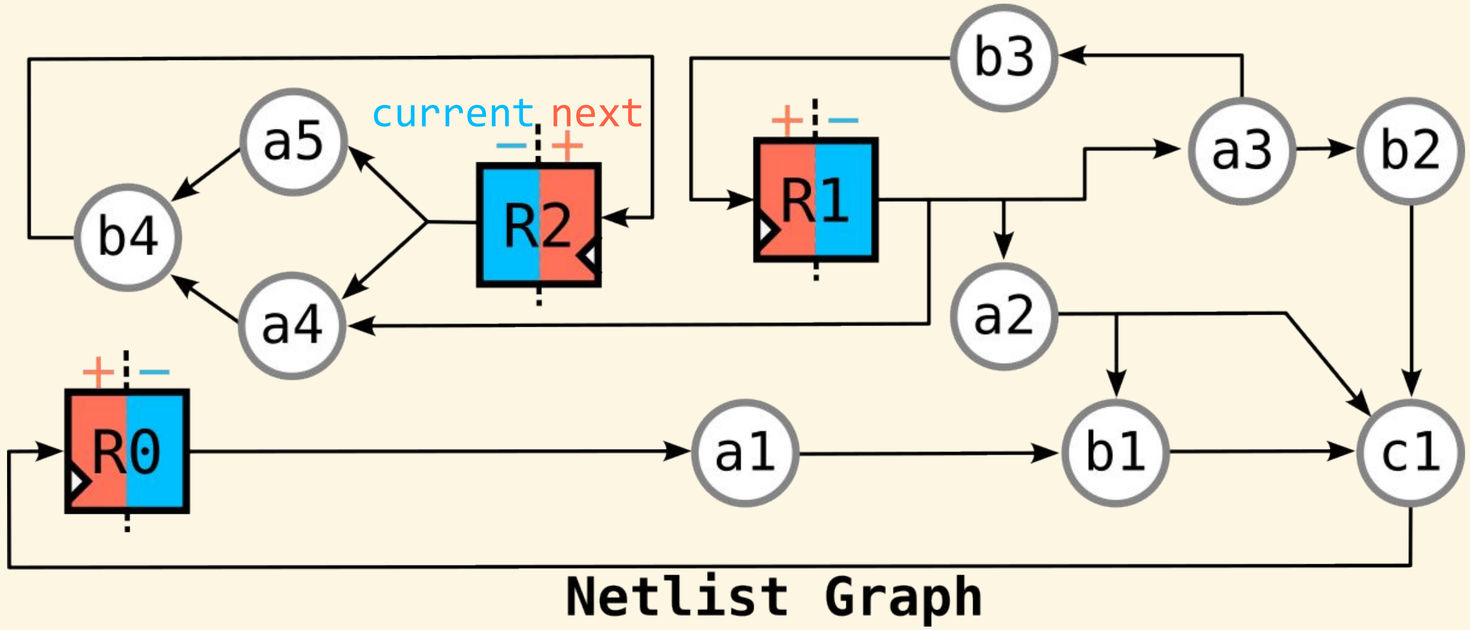
- A *netlist* → a cyclic graph
- “Stateless” combinational logic
- “Stateful” registers and memories
  - Retain values at clock cycle



Cycle-accurate RTL Simulation*

Split registers

- current and next value



Netlist Graph

* Not timing accurate, but good enough for almost all **functional** verification tasks.

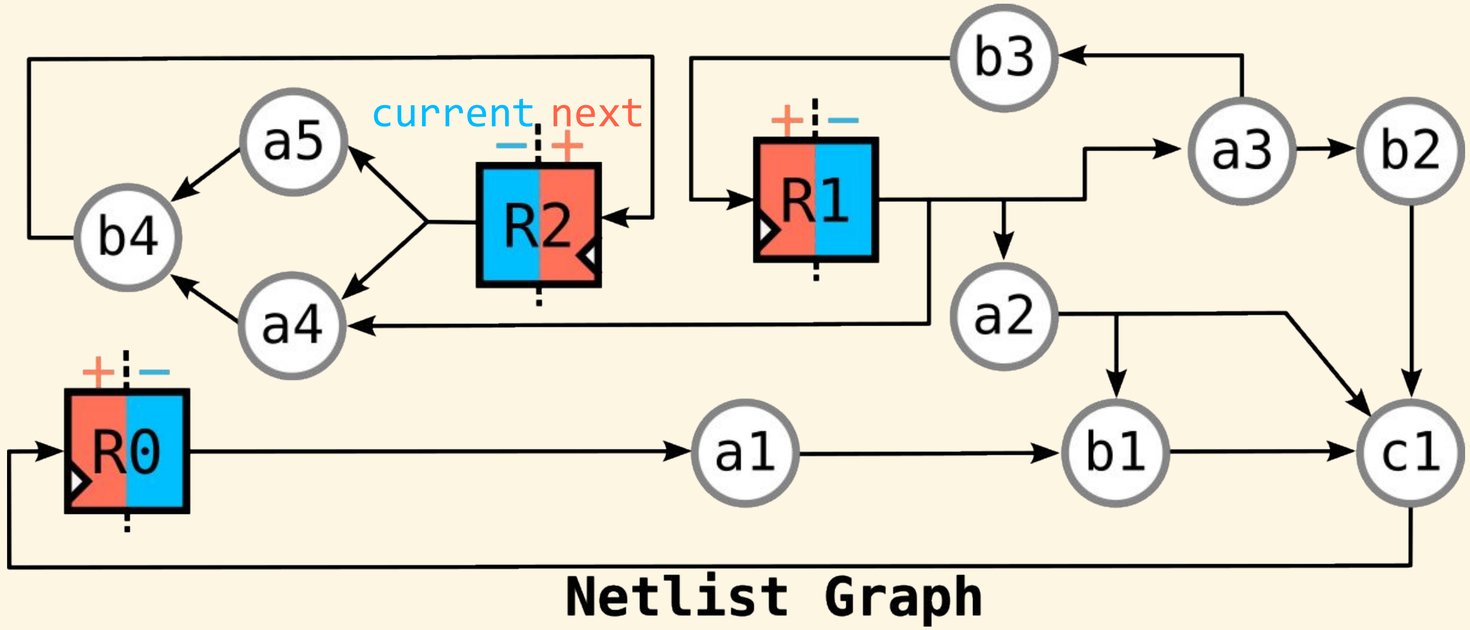
Cycle-accurate RTL Simulation*

Split registers

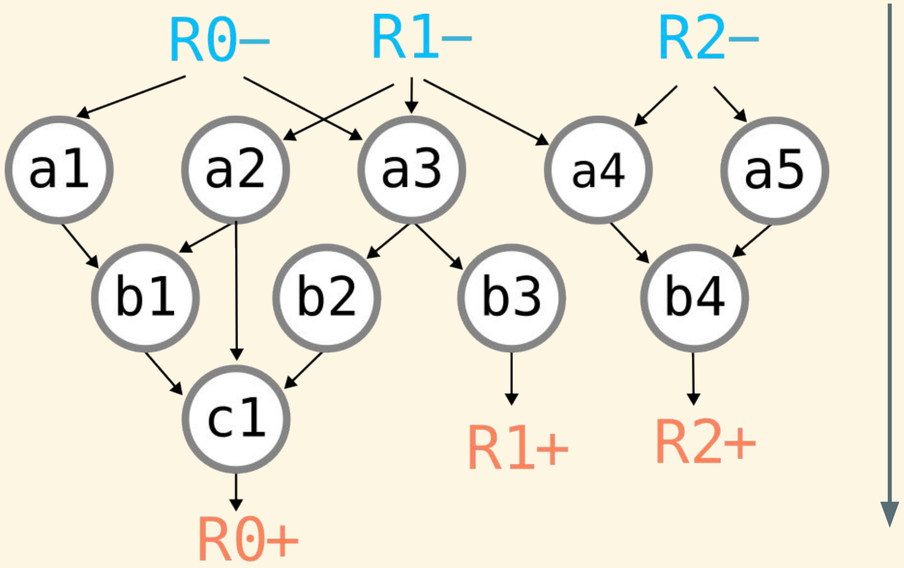
- current and next value

At each cycle:

- Compute next
- Overwrite current
- Repeat



Netlist Graph



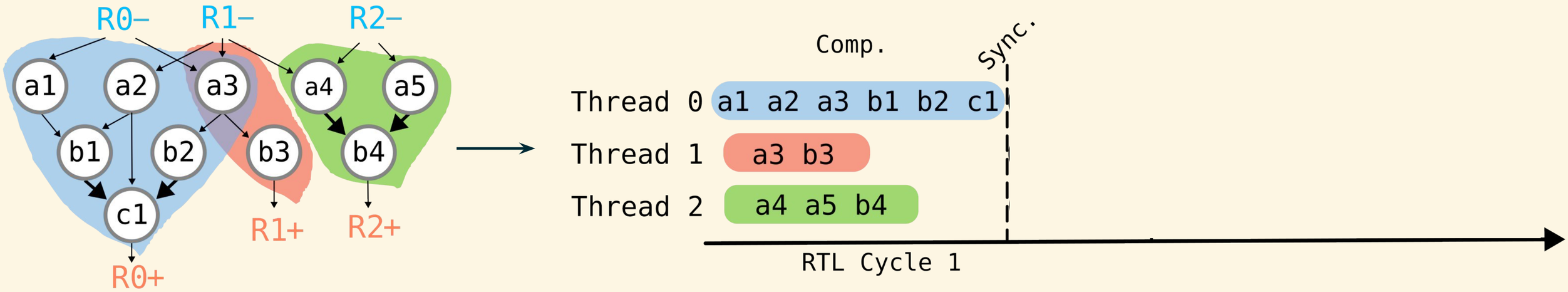
Data Dependence Graph

* Not timing accurate, but good enough for almost all **functional** verification tasks.

Bulk-synchronous Parallel Simulation (BSP)

Computation phase

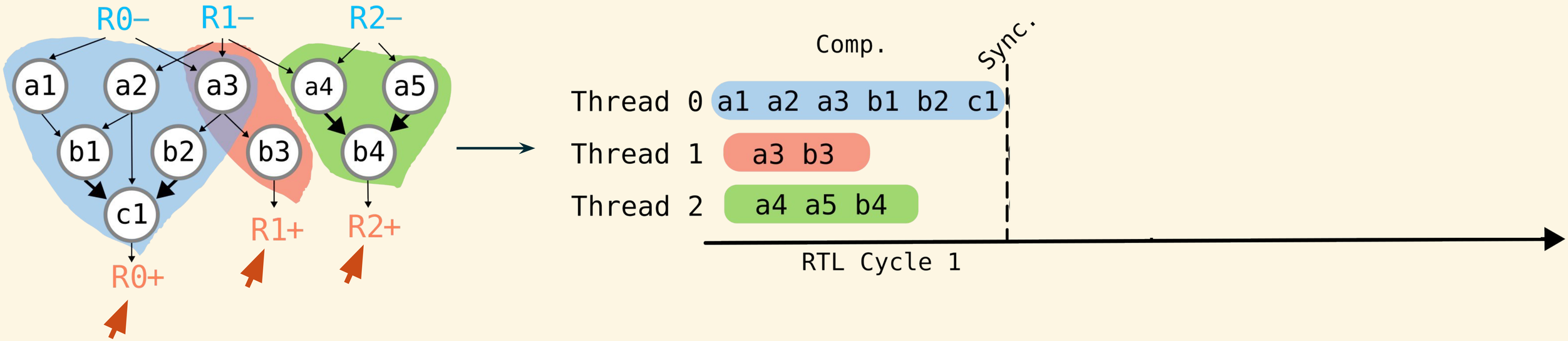
- Independently compute program partitions
- Globally synchronize



Bulk-synchronous Parallel Simulation (BSP)

Computation phase

- Independently compute program partitions
- Globally synchronize



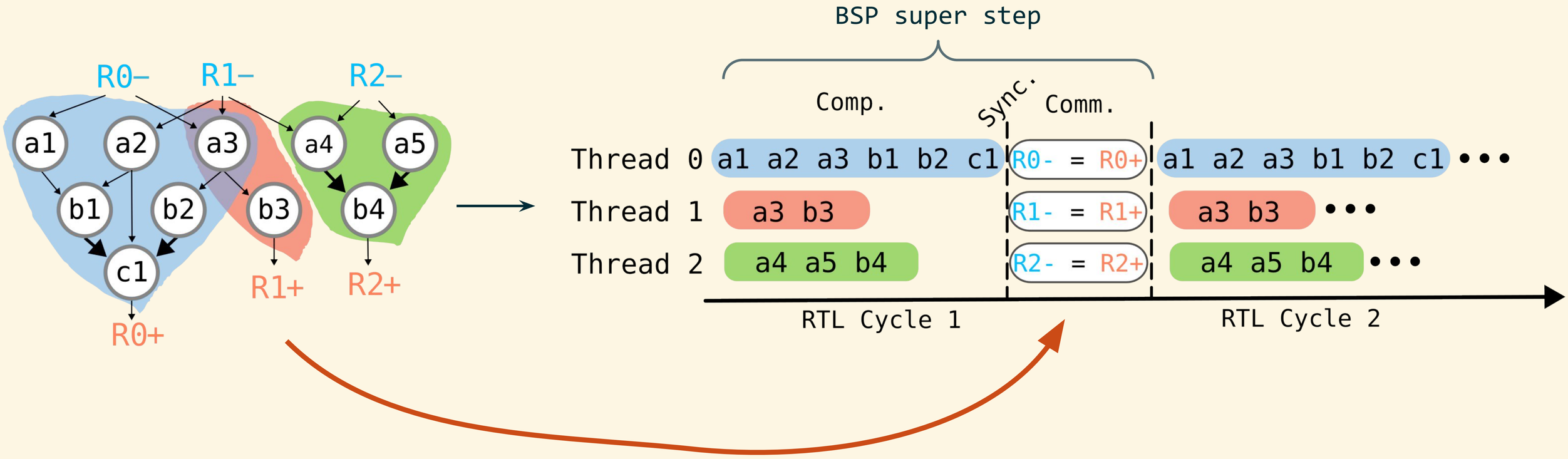
Bulk-synchronous Parallel Simulation (BSP)

Computation phase

- Independently compute program partitions
- Globally synchronize

Communication phase

- Exchange produced values
- Globally synchronize



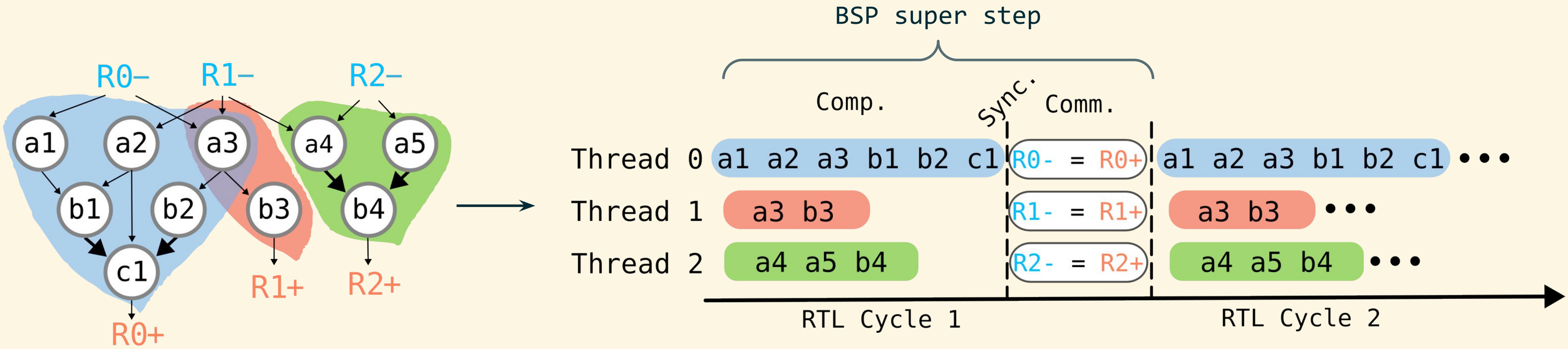
Bulk-synchronous Parallel Simulation (BSP)

Computation phase

- Independently compute program partitions
- Globally synchronize

Communication phase

- Exchange produced values
- Globally synchronize



How does this scale to 1000s of cores/threads?

Thousand-way Parallel Simulation

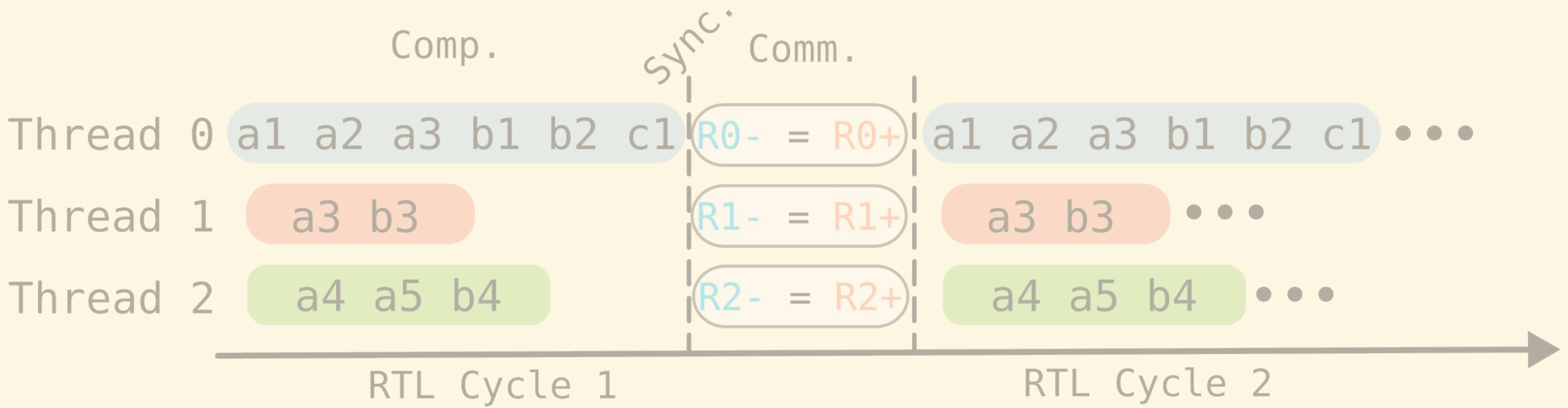
Scaling Simulation Performance

number of cores

$$r(N) = \frac{1}{t_{comp} + t_{sync} + t_{comm}}$$

simulation rate

Minimize



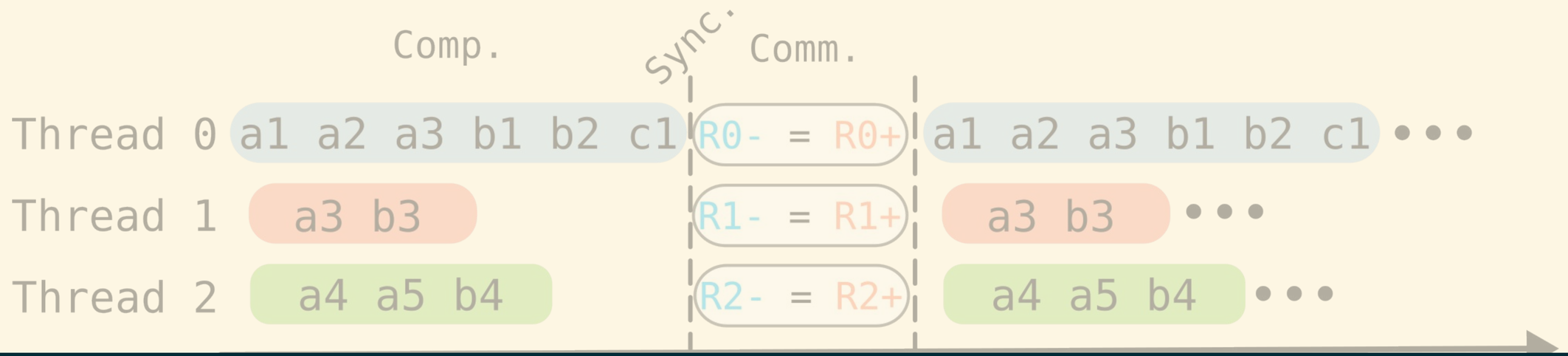
Scaling Simulation Performance

number of cores

$$r(N) = \frac{1}{t_{comp} + t_{sync} + t_{comm}}$$

simulation rate

Minimize



Why use the IPU?
What about a general purpose processor?

IPU or x64

$$r(N) = \frac{1}{t_{comp} + t_{sync} + t_{comm}}$$

	computation	synchronization	communication
IPU	1000s of slow cores X	≈const ✔	On-chip ≈ const ✔ Off-chip varies X
x64	10s/100s of fast cores ✔	increasing with N X	increases with N X

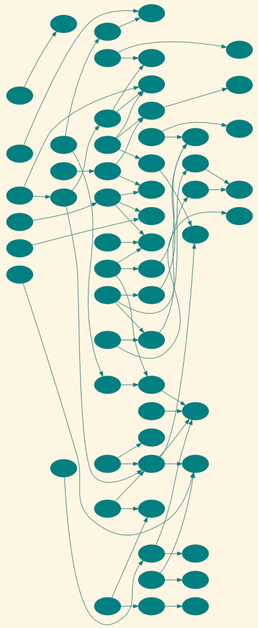
IPU or x64

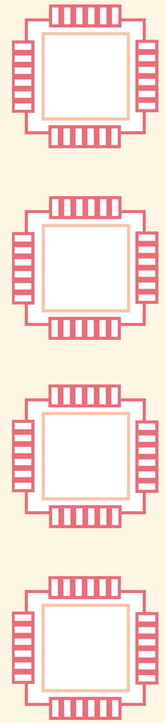
$$r(N) = \frac{1}{t_{comp} + t_{sync} + t_{comm}}$$

	computation	synchronization	communication
IPU	1000s of slow cores X	≈const ✔	On-chip ≈ const ✔ Off-chip varies X
x64	10s/100s of fast cores ✔	increasing with N X	increases with N X

See paper for the quantitative treatment

Overview of Parendi



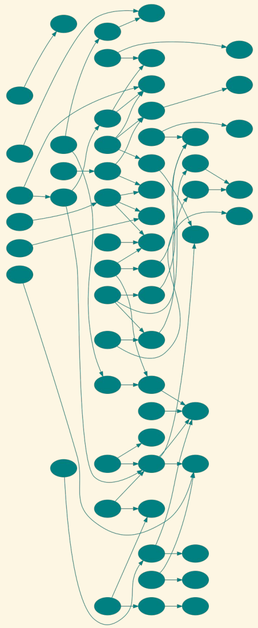


DAG

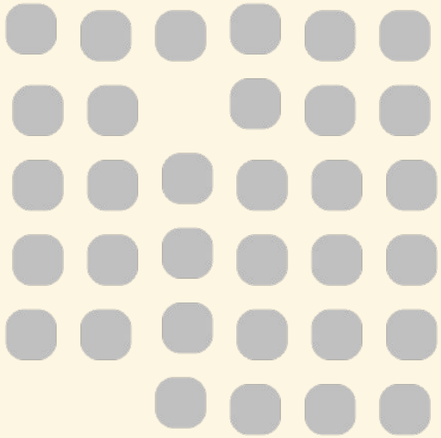
Tiles

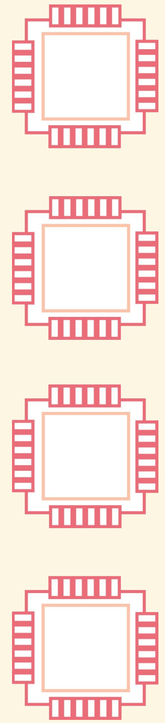
1. Build a data dep. graph from source code (Verilog)

Overview of Parendi



extract parallelism

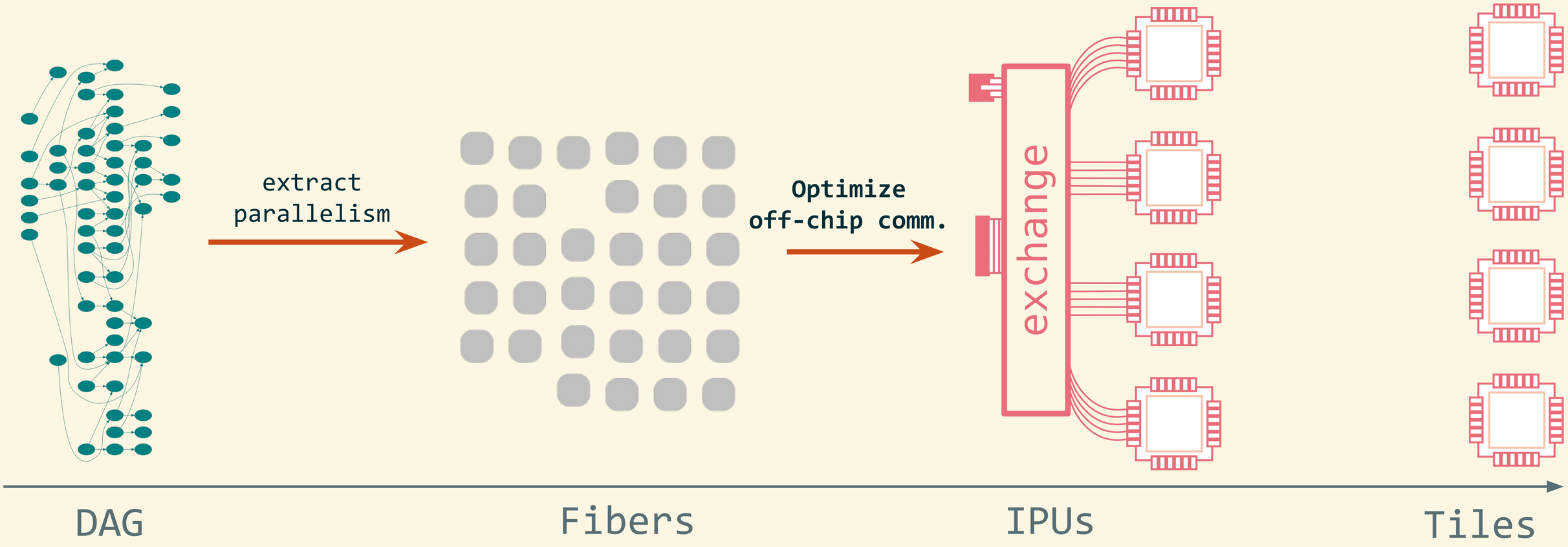




DAG Fibers Tiles

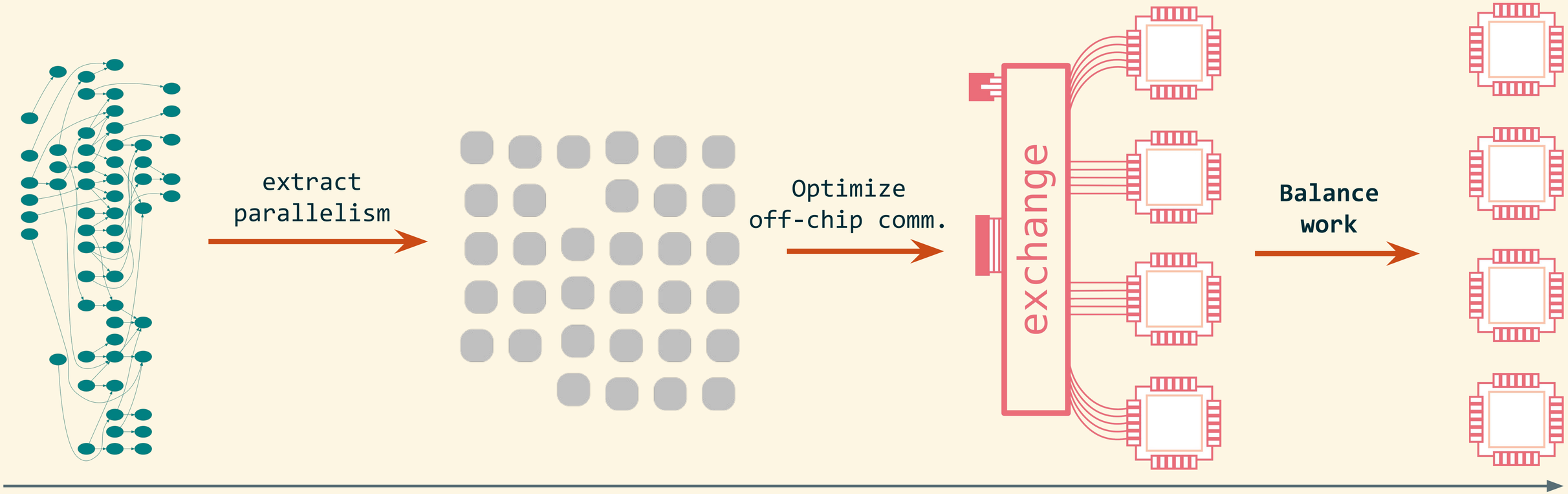
1. Build a data dep. graph from source code (Verilog)
2. **Extract *fibers* → smallest units of independent work**

Overview of Parendi



1. Build a data dep. graph from source code (Verilog)
2. Extract *fibers* → smallest units of independent work
3. **Partition fibers across IPUs → hypergraph partitioning**

Compiling RTL to the IPU

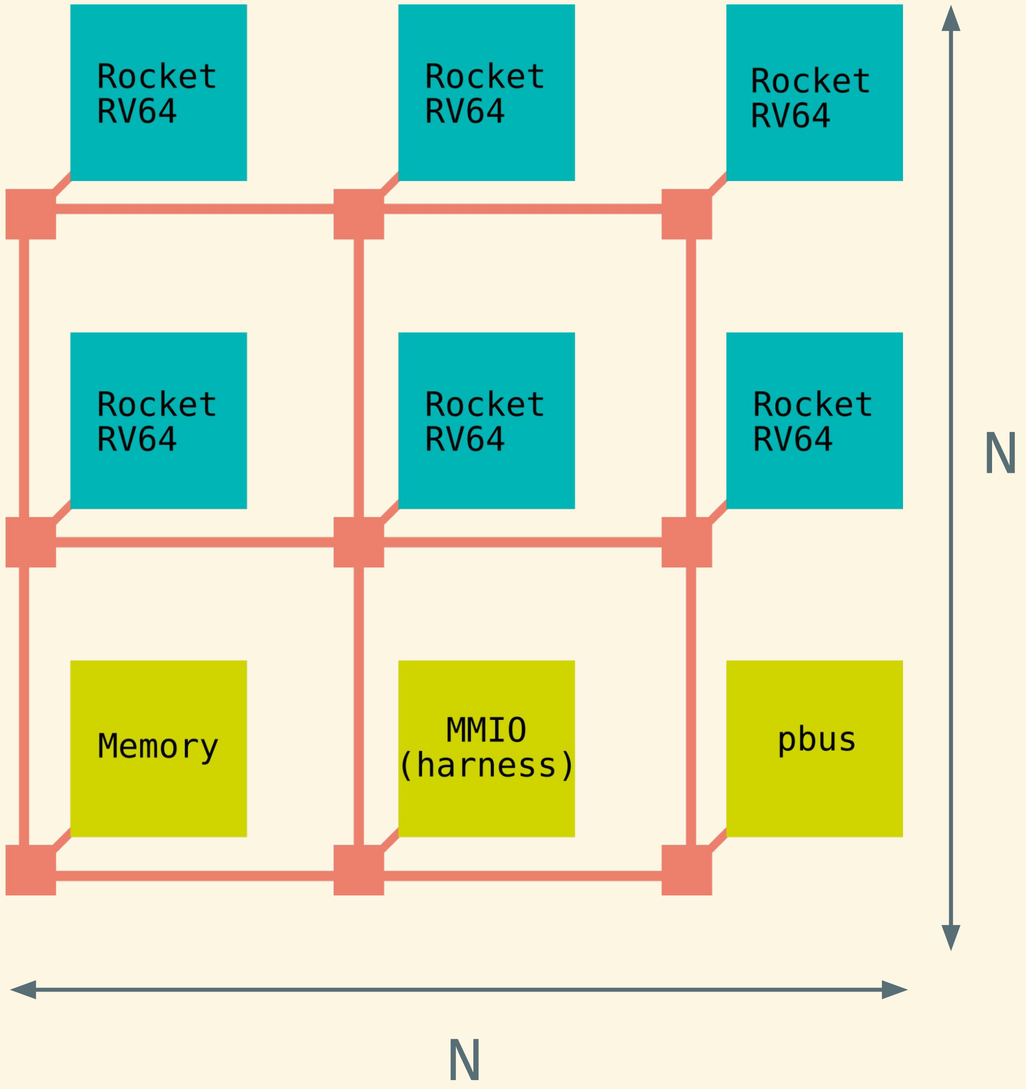


1. Build a data dep. graph from source code (Verilog)
2. Extract *fibers* → smallest units of independent work
3. Partition fibers *across* IPUs → hypergraph partitioning
4. **Partition fibers *within* IPUs → submodular load balancing**

Evaluation

Evaluation

- {l|s}r**N** → RV64 rocket[1] cores on NxN mesh[2]
- lr → Large rocket cores
- sr → Small rocket cores
- Light-weight Verilog harness



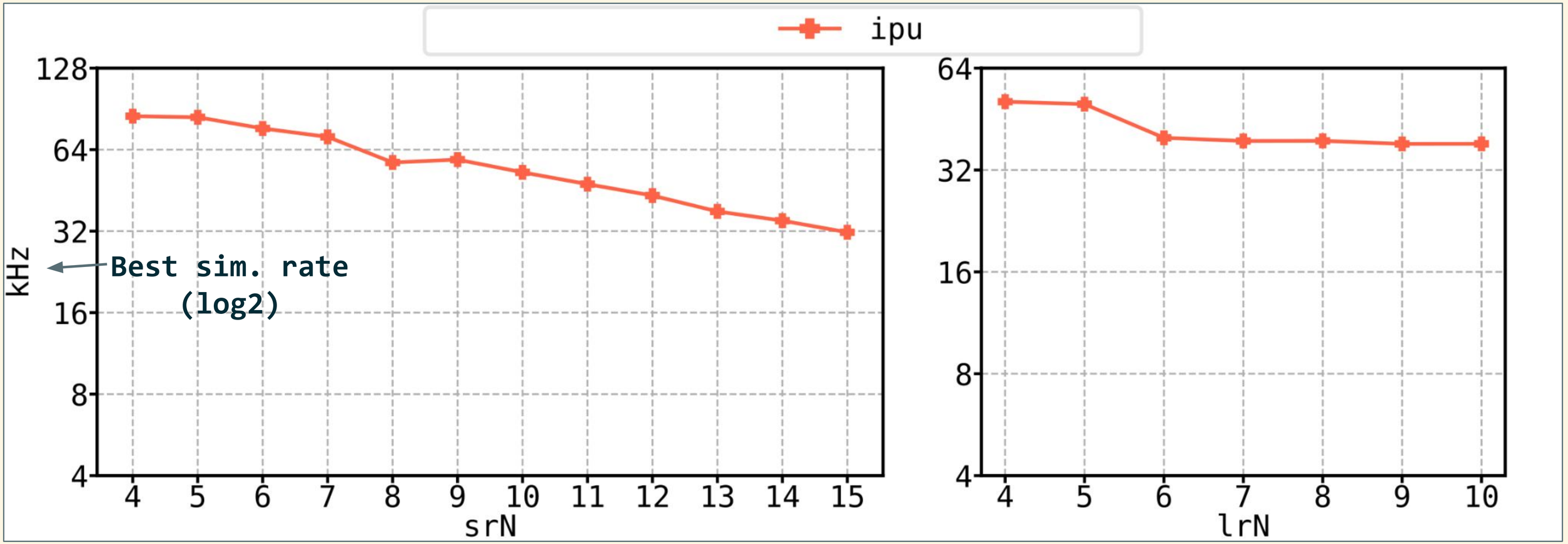
[1] Krste Asanović et al, “The Rocket Chip Generator.” (2016).
[2] Jerry Zhao et al, "Constellation: An Open-Source SoC-Capable NoC Generator," 2022 15th IEEE/ACM International Workshop on Network on Chip Architectures (NoCArc 2022).

Hardware Setup

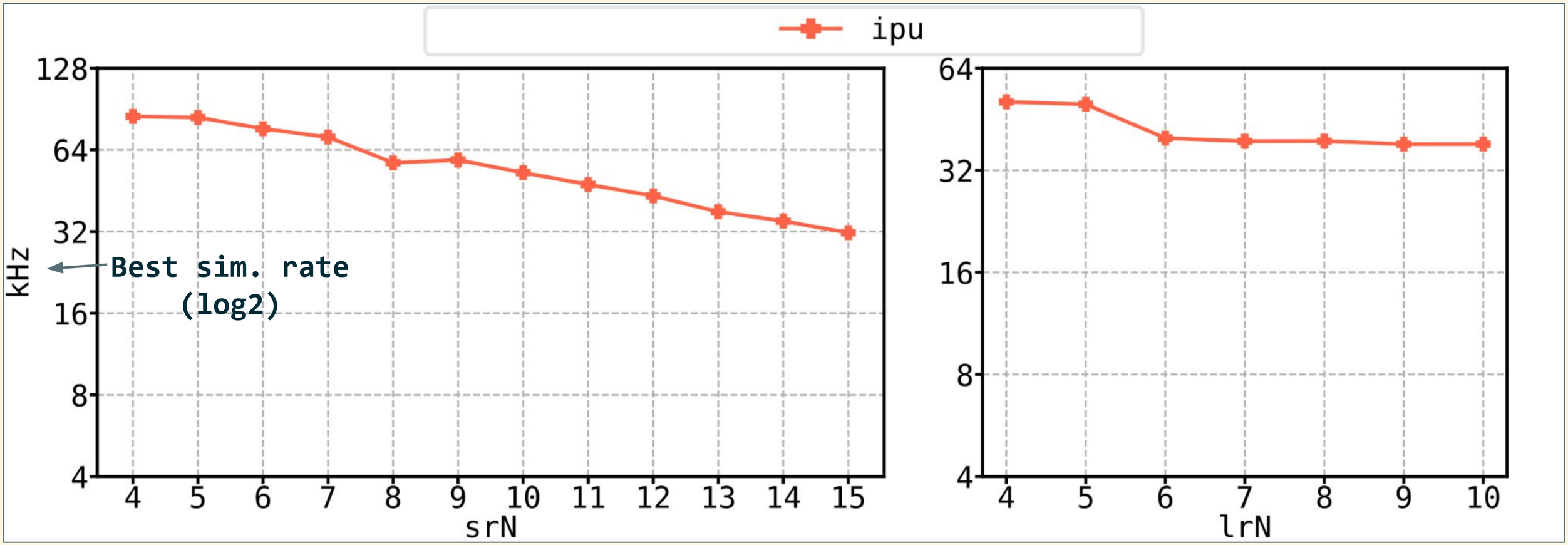
	Verilator (v5.006)		Parendi
Hardware	AMD EPYC 9954 (**ae4**)	Intel Xeon 6348 (**ix3**)	4xIPU M2000 (**ipu**)
# cores avail.	64x2	28x2	1472x4
# cores used	1, 2, 4, 6,..., 32	1, 2, 4, 6,..., 32	1472, 2944, 4416, 5888
Freq. GHz	3.75	3.5	1.35
SRAM (MiB)	386x2	79x2	900x4
Released	Q4 2022	Q2 2021	Q3 2020

Verilator parallelism limited to 32 cores due to excessive compile time (+8hr); see paper.

Simulation Rates



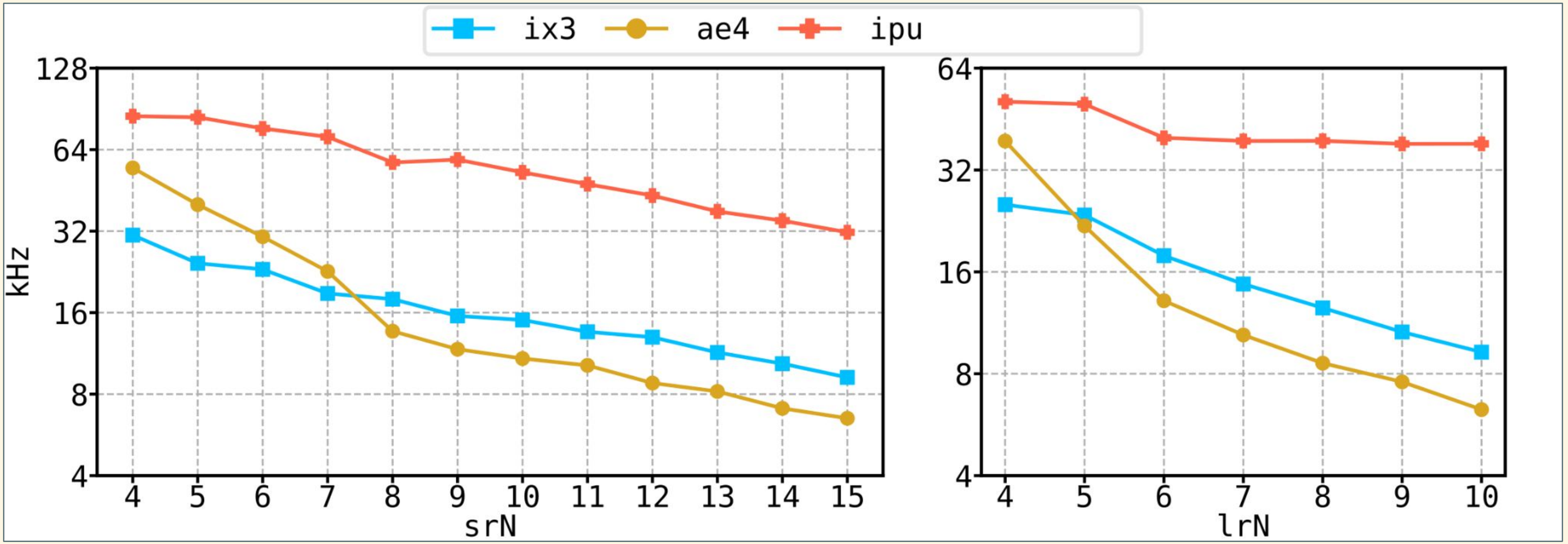
Simulation Rates



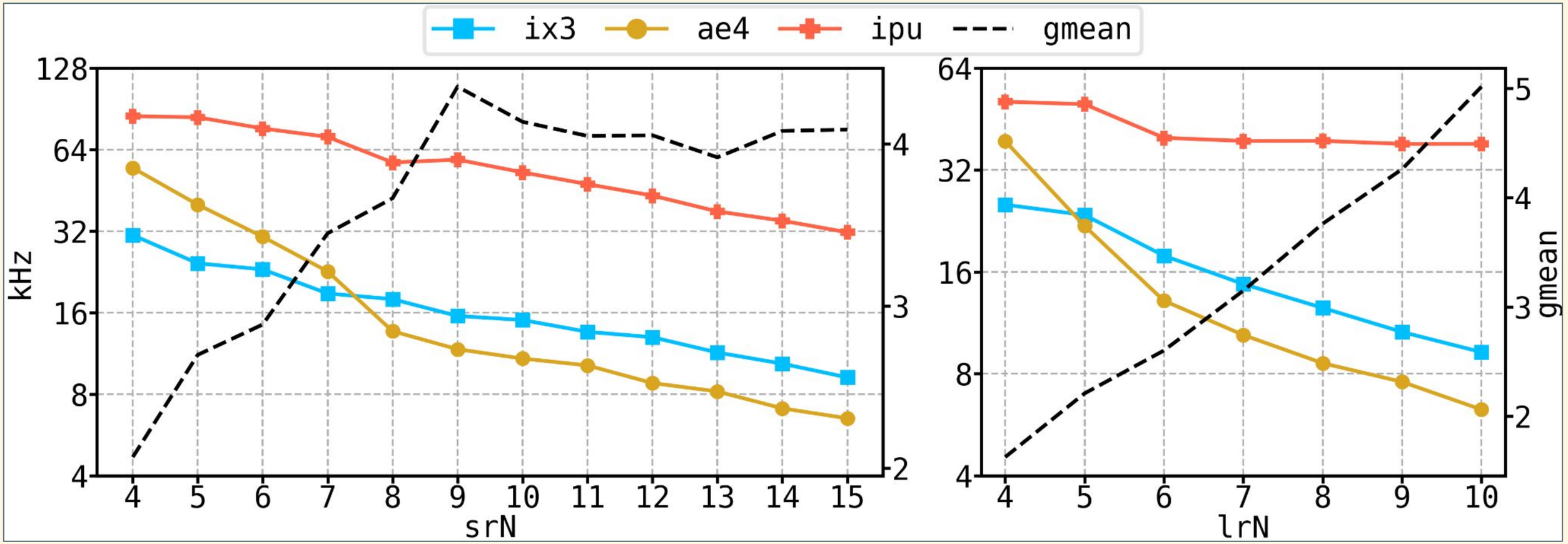
increasing design size
small rocket cores

increasing design size
large rocket cores

Simulation Rates



Simulation Rates



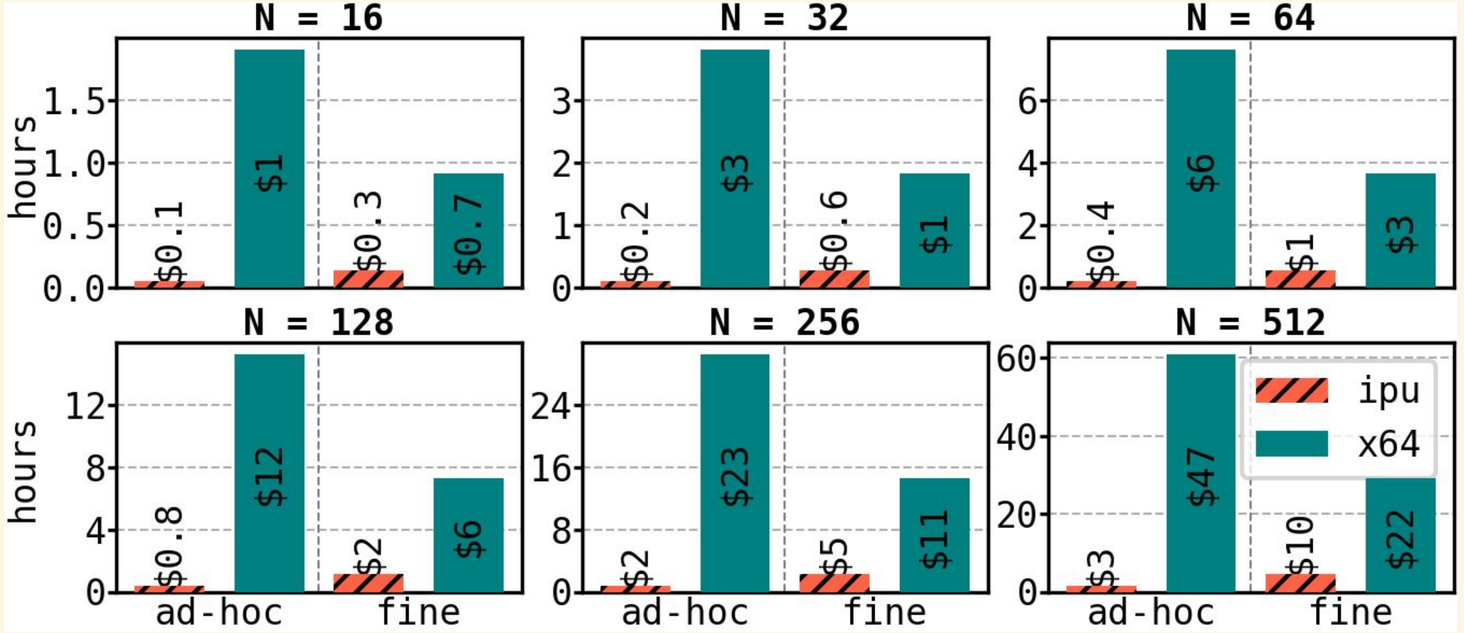
Conclusion

- Thousand-way parallel RTL simulation is becoming a necessity
- Parendi shows thousand-way parallelism is feasible
- Runs on real hardware → Graphcore IPU
- **5888** cores across 4 sockets on a server board
- Multi-stage parallelization algorithm
- Up to 4x faster than Verilator on high-end x64
  - See paper for more evaluation, including cost, other academic work, microbenchmarks and etc.

Backup

Cost Analysis

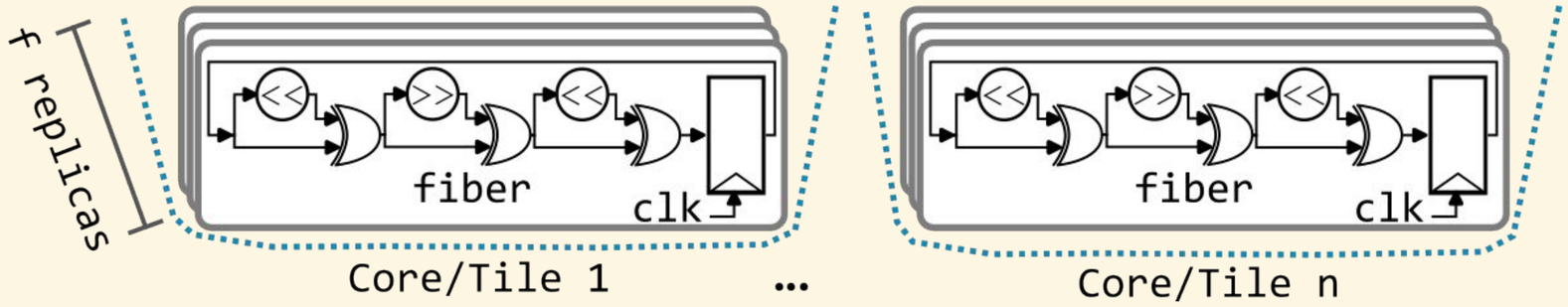
- Regression testing scenario
- N identical tests
  - short → 1 million cycles
- Ad-hoc:
  - IPU: One test per IPU (1472 tiles)
  - x64: One test per CPU core
- Fine:
  - Tests parallelized to multiple optimal (perf.) tiles and cores



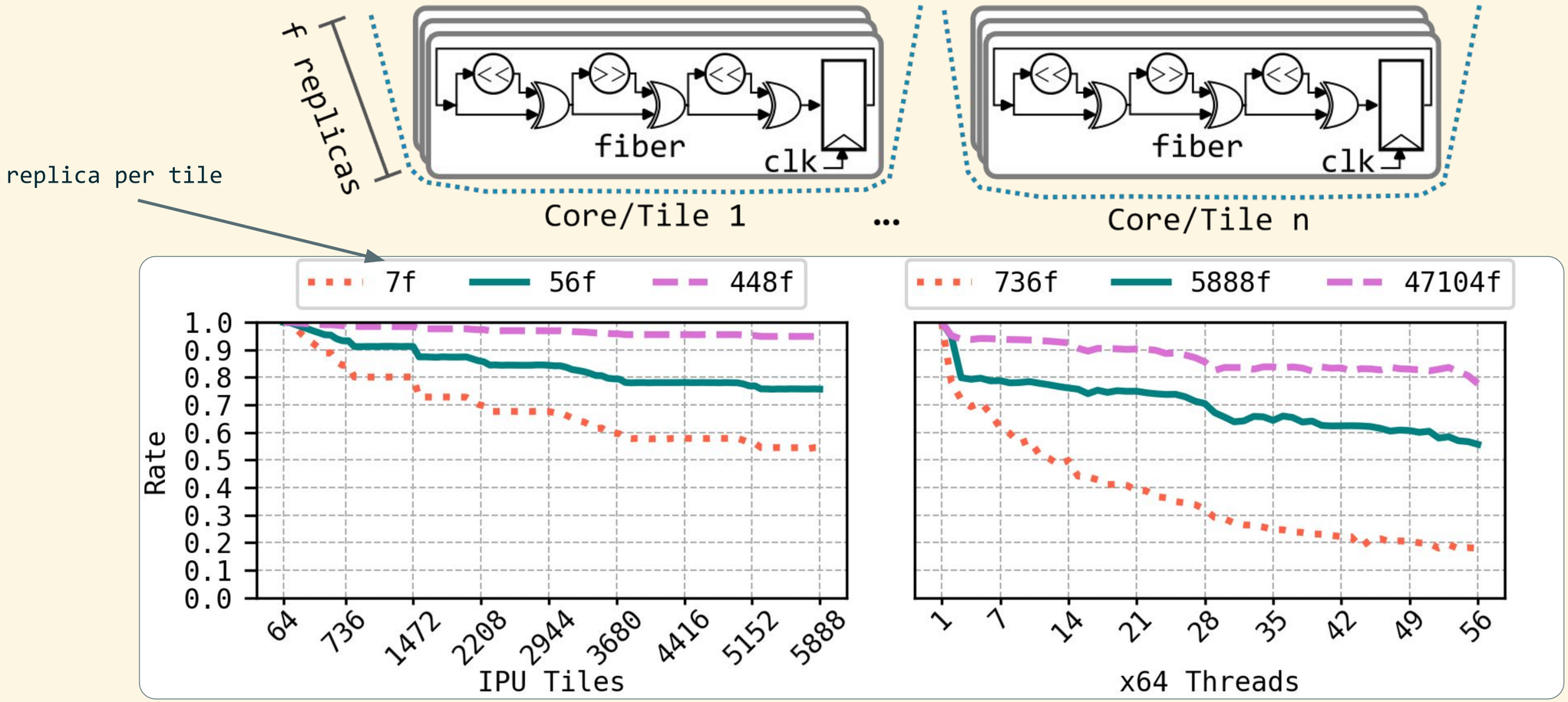
Weak Scaling

$$r(N) = \frac{1}{t_{comp} + t_{sync} + t_{comm}}$$

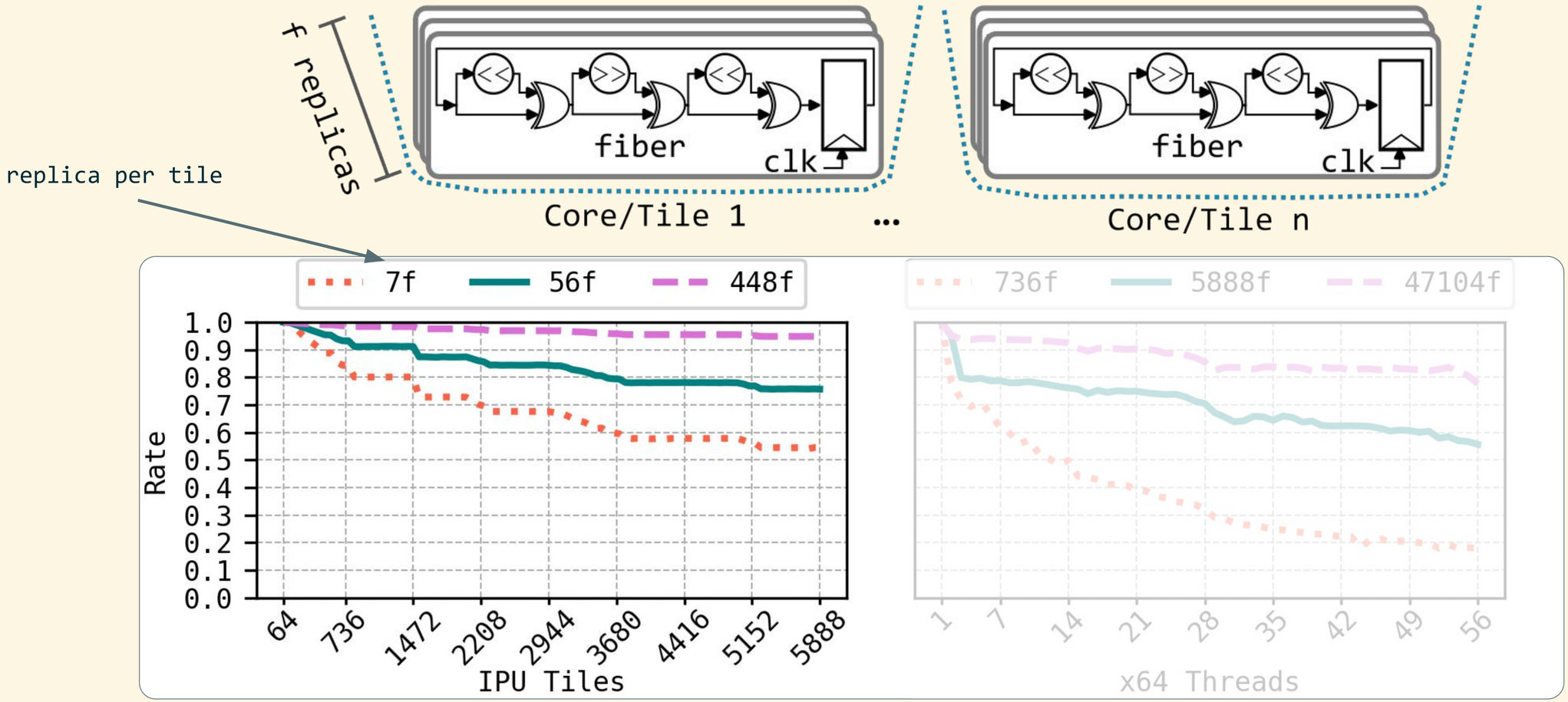
- Increase *total* work and parallelism proportionally
  - Computation time remains constant w.r.t to N
- Simplification → no communication cost
- Weak scaling goal → constant sim. rate w.r.t to N



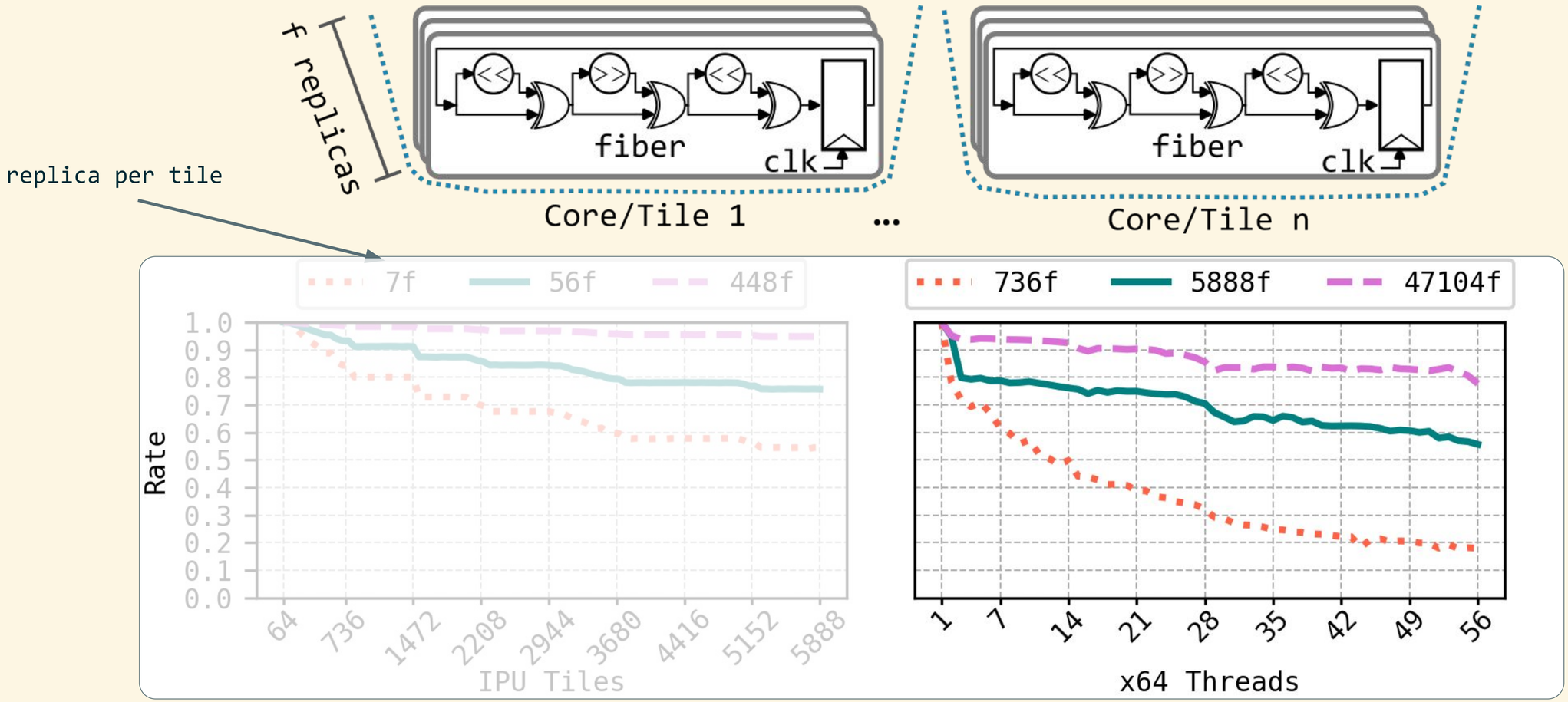
Sync. Cost Analysis



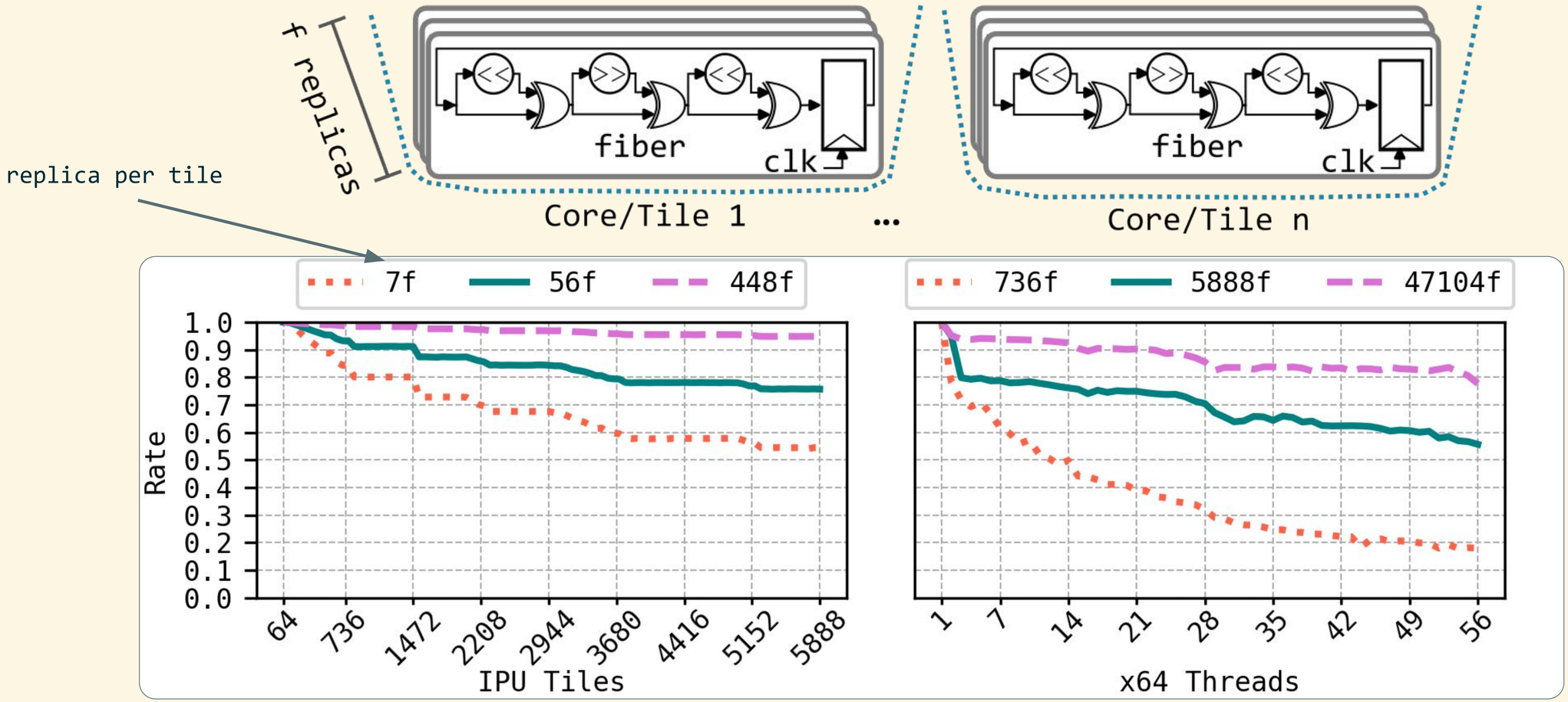
Sync. Cost Analysis



Sync. Cost Analysis



Sync. Cost Analysis



The IPU has far more scalability potentials: sync. cost is not an issue!