# Cyrus: Unintrusive Application-Level Record-Replay for Replay Parallelism

Nima Honarmand, Nathan Dautenhahn,
Josep Torrellas and Samuel T. King (UIUC)
Gilles Pokam and Cristiano Pereira (Intel)


iacoma.cs.uiuc.edu

ILLINOIS
illinois.edu

i-acoma group

# Record-and-Replay (RnR)

- Record execution of a parallel program or a whole machine
  - Save non-deterministic events in a log
- During replay, use the recoded log to enforce the same execution
  - Each thread follows the same sequence of instructions
- Use cases
  - Debugging
  - Security
  - High availability

# Contribution: Cyrus RnR System

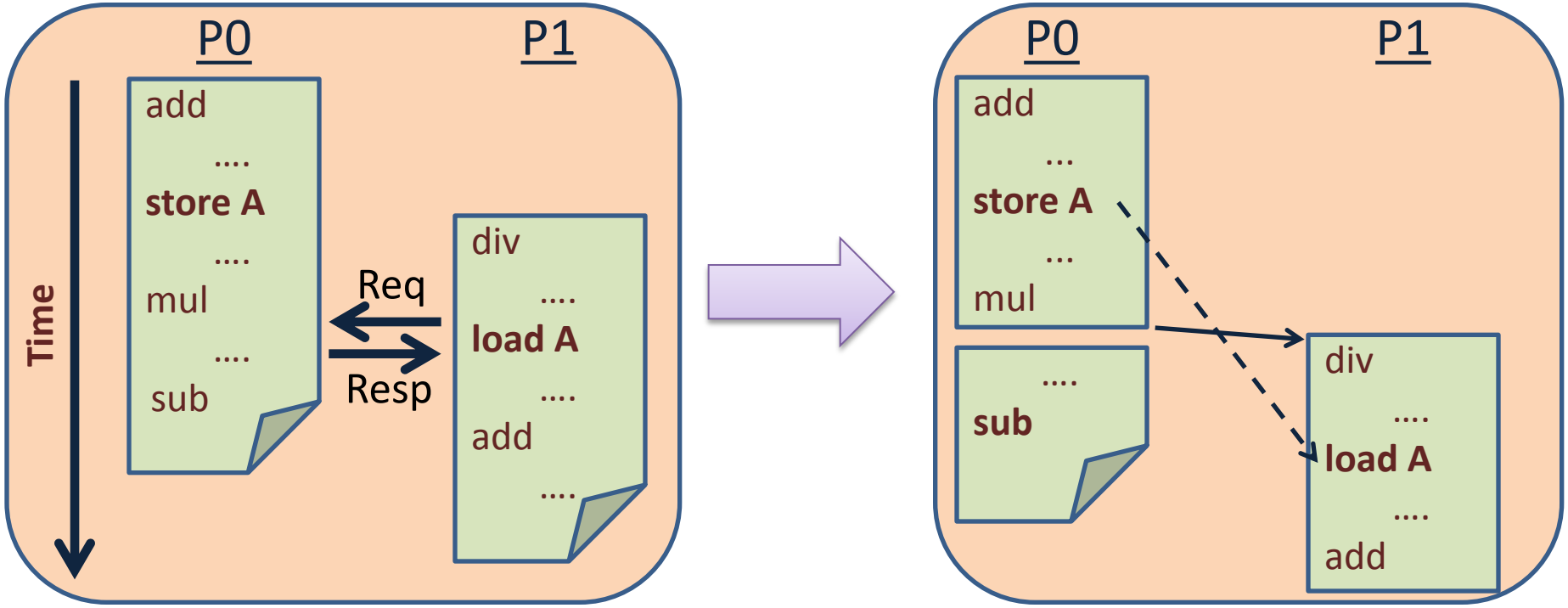
- Application-level RnR
  - RnR one or more programs in isolation
  - What users typically need
- Fast replay
  - Replay-time parallelism
  - Flexibly trade off parallelism for log size
- Unintrusive HW
  - No changes to snoopy cache coherence protocol

# Capturing Non-determinism

- Sources of non-determinism
  - Program inputs
  - Memory access interleavings
- How to capture?
  - OS kernel extension to capture program inputs
  - HW support to capture memory interleavings (HW-assisted RnR)
- This talk: recording memory interleavings

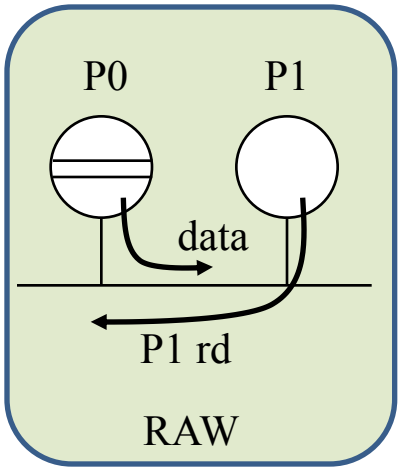
# Recording Interleaving as Chunks

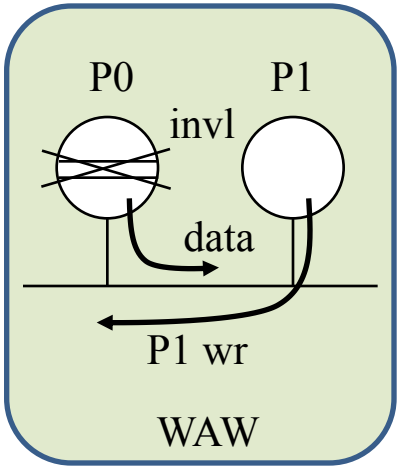
- Inter-processor data dependences manifest as coherence messages
- Capture interleavings as ordered chunks of instructions

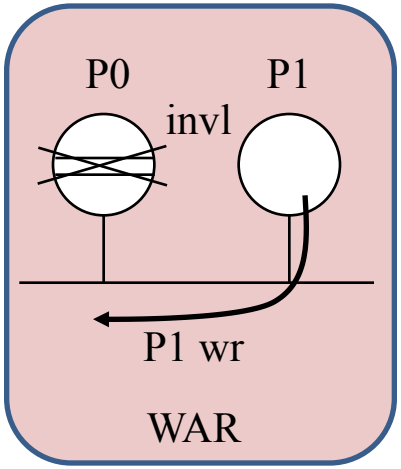


# Restriction: Unintrusive HW

- Unmodified snoopy protocols
  - In some coherence transactions, there is no reply



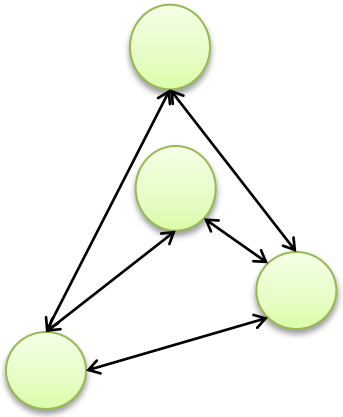

- Requirements for HW-assisted RnR:
  - Do not augment or add coherence messages
  - Do not rely on explicit replies

Only source is always aware → Use *source-only* recording

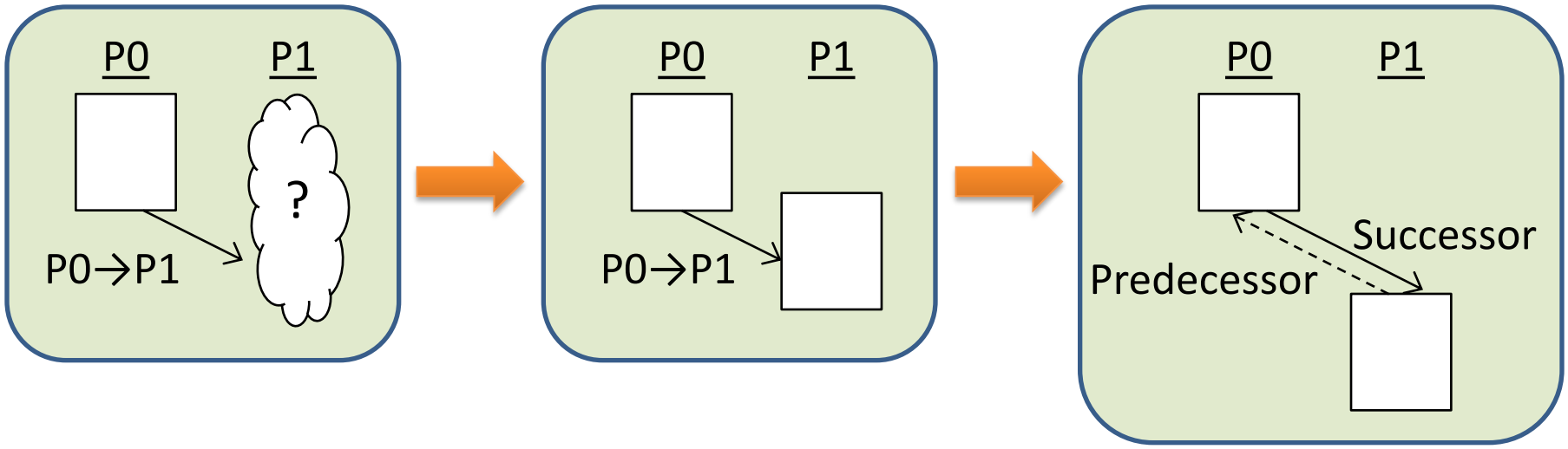
# Challenge 1: Enable Replay Parallelism

- Key to fast replay
  - Overlapped replay of chunks from diff. threads

- Previous work:
  - DAG-based ordering (Karma [ICS'2011])
    - Requires explicit replies
    - Augments coherence messages



# Challenge 1: Enable Replay Parallelism

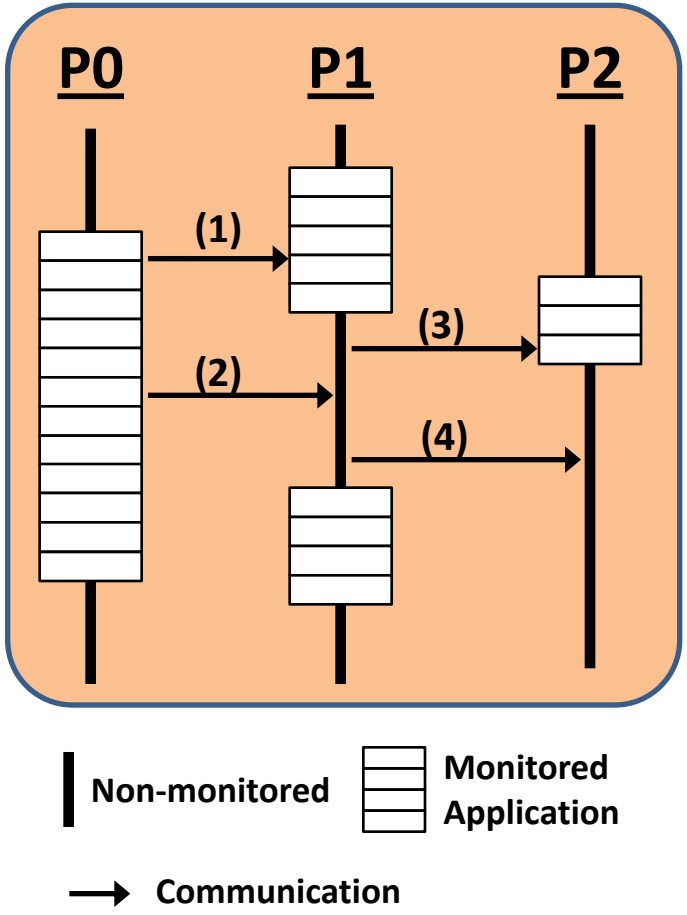


# Challenge 2: Application-Level RnR

Turn hardware on only when a recorded application runs.

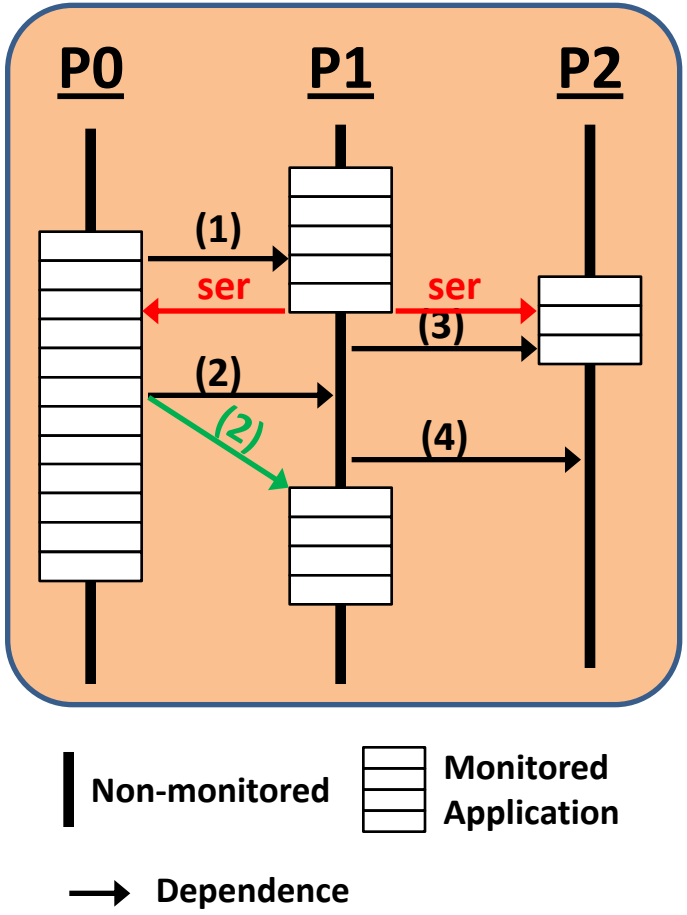
Four cases:
(1) src=monitoring, dst=monitoring
(2) src=monitoring, dst=not monitoring
(3) src=not monitoring, dst=monitoring
(4) src=not monitoring, dst=not monitoring

Issues of source-only recording:
- Cannot distinguish between (1) and (2)
- (2) may result in a dependence later
- Not recording in (3) and (4)

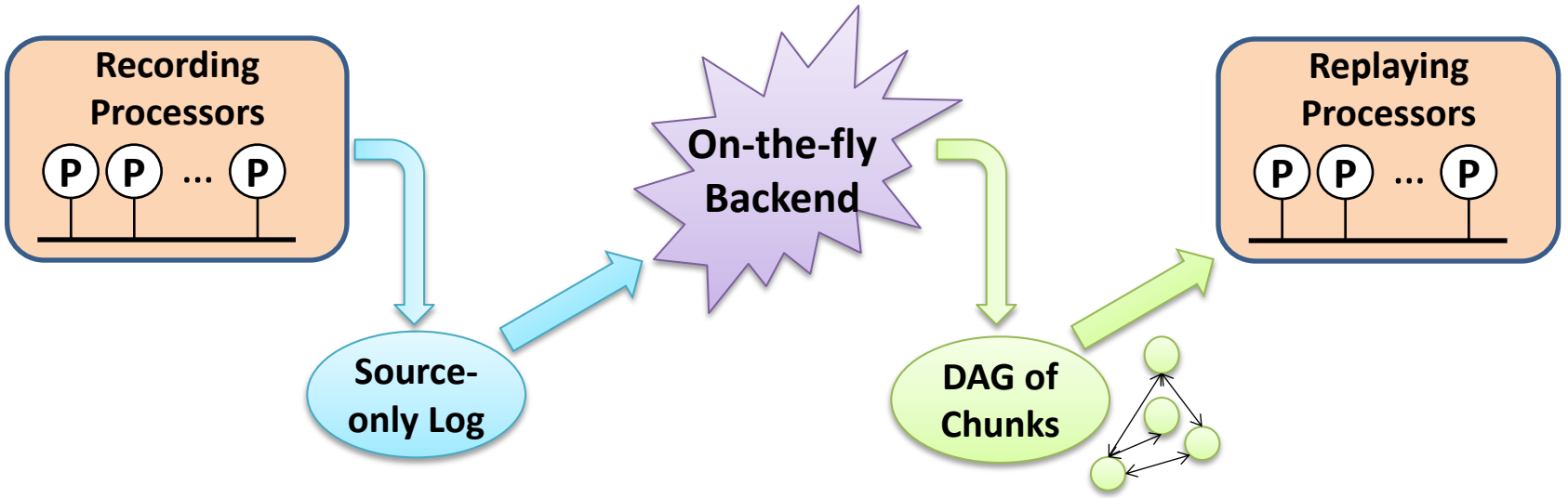


# Challenge 2: Application-Level RnR

- Treat (2) as an Early Dependence
  - Defer and assign it to the next chunk of the target processor
- (3) and (4) superseded by context switches
  - At context switch, record a Serialization Dependence to all other processors



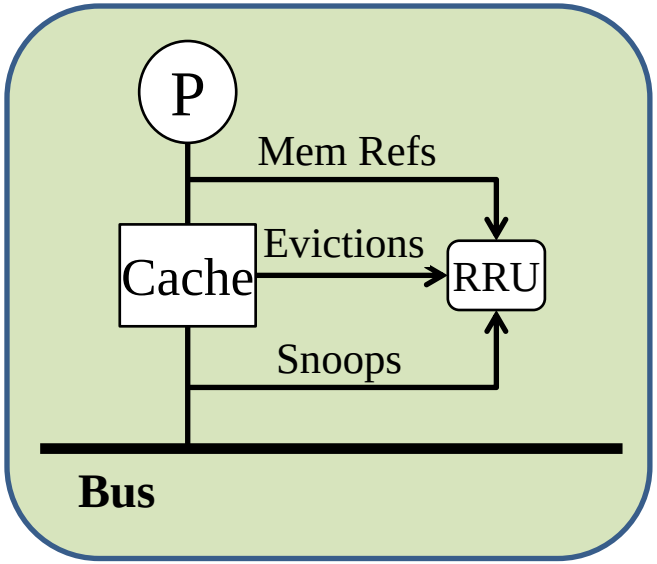
# Key: On-the-Fly Backend Software Pass



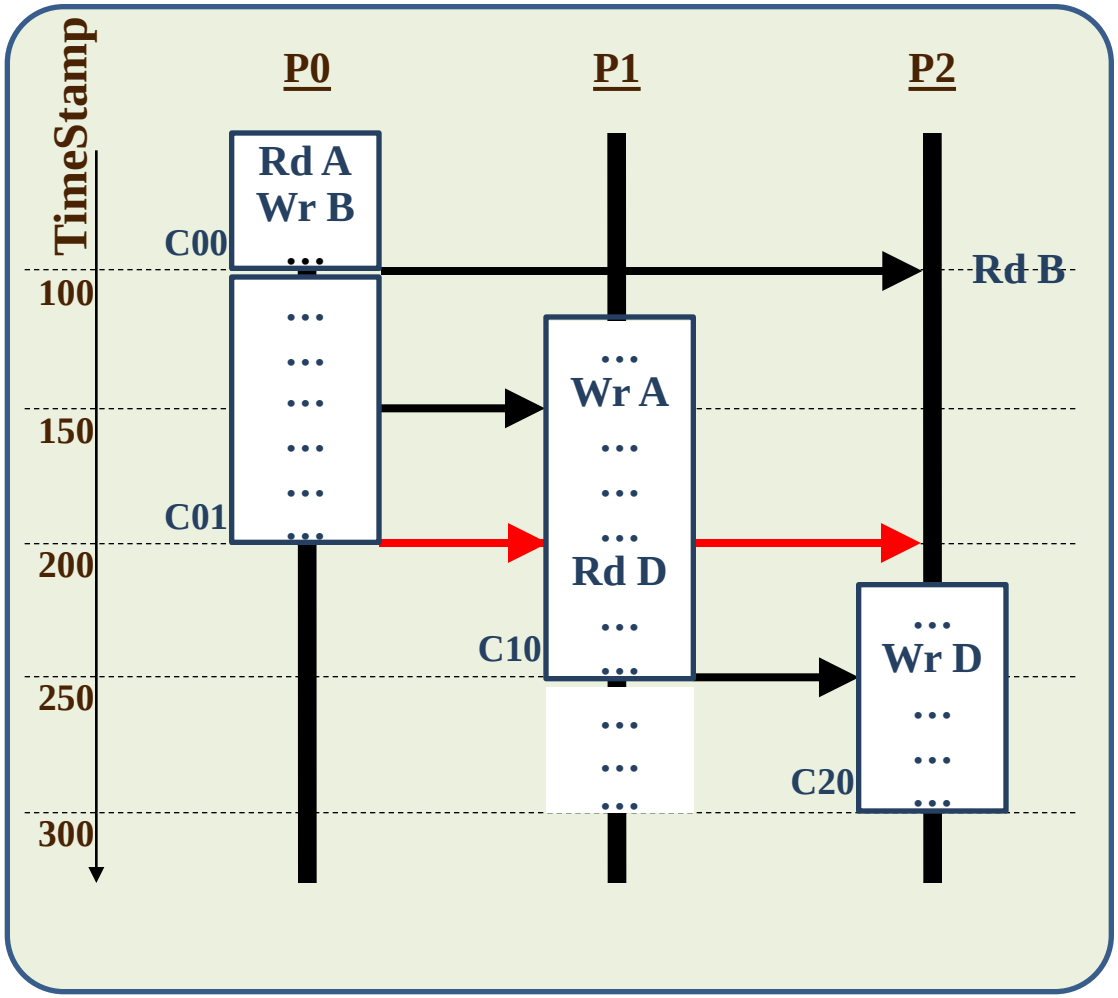

- Transforms source-only log to DAG (for parallelism)
- Fixes the Early and Serialization dependences
  - To support app-level RnR
- Can trade replay parallelism for log size

# Memory Race Recording Unit (RRU)

- HW module that observes coherence transactions and cache evictions
- Tracks loads/stores of the chunk in a signature
- Keeps signatures for multiple recent chunks
- Records for each chunk
  - # of instructions
  - Timestamp (# of coh. transactions)
  - Dependences for which the chunk is source
- Dumps recorded chunks into a log in memory



# RRUs Record Source-Only Log

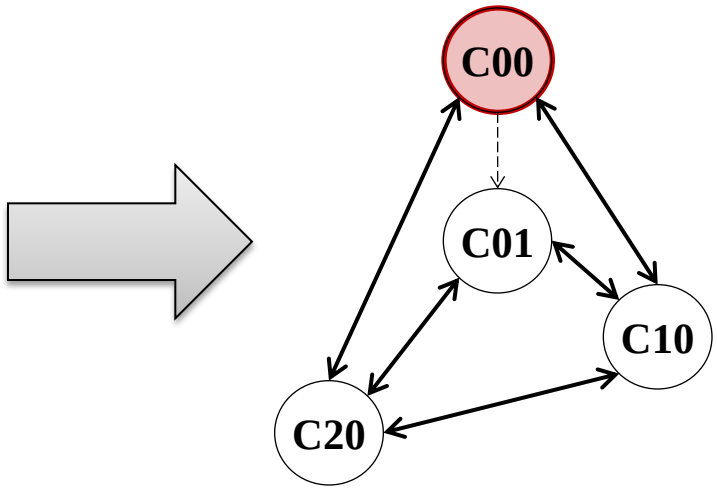



| | Chunk TS | Successor Vector | | |
|---|---|---|---|---|
| | | P0 | P1 | P2 |
| **P0** | | | | |
| C00 | 100 | - | 150 | 100 |
| C01 | 200 | - | 200 | 200 |
| **P1** | | | | |
| C10 | 250 | - | - | 250 |
| **P2** | | | | |
| C20 | 300 | - | - | - |

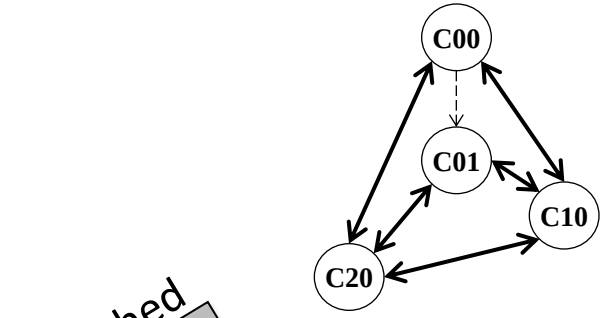
# Backend Pass Creates DAG

- Finds the target chunk for each recorded dependency
  - Creates bidirectional links between src and dst chunks
- This algorithm is called MaxPar

| | | | | | |
|---|---|---|---|---|---|
| Chunks of P0 | C00 | 100 | - | 150 | 100 |
| | C01 | 200 | - | 200 | 200 |
| Chunks of P1 | C10 | 250 | - | - | 250 |
| Chunks of P2 | C20 | 300 | - | - | - |

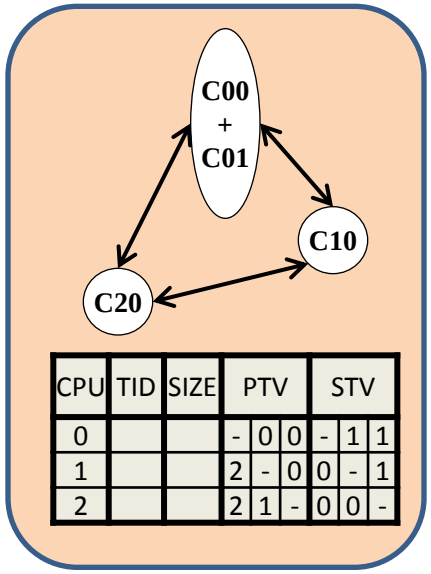


# Trading Replay Parallelism for Log Size



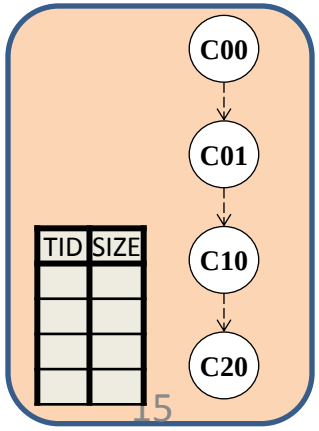

| CPU | TID | SIZE | PTV | | | STV | | |
|---|---|---|---|---|---|---|---|---|
| 0 | | | - | 0 | 0 | - | 1 | 1 |
| 0 | | | - | 0 | 0 | - | 1 | 1 |
| 1 | | | 2 | - | 0 | 0 | - | 1 |
| 2 | | | 2 | 1 | - | 0 | 0 | - |

Stitched



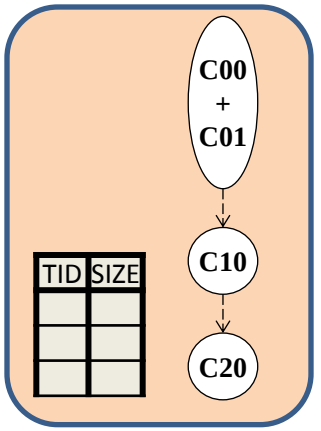

- Less Parallelism
- Smaller log

Serial




- No Parallelism
- Even Smaller log

StSerial

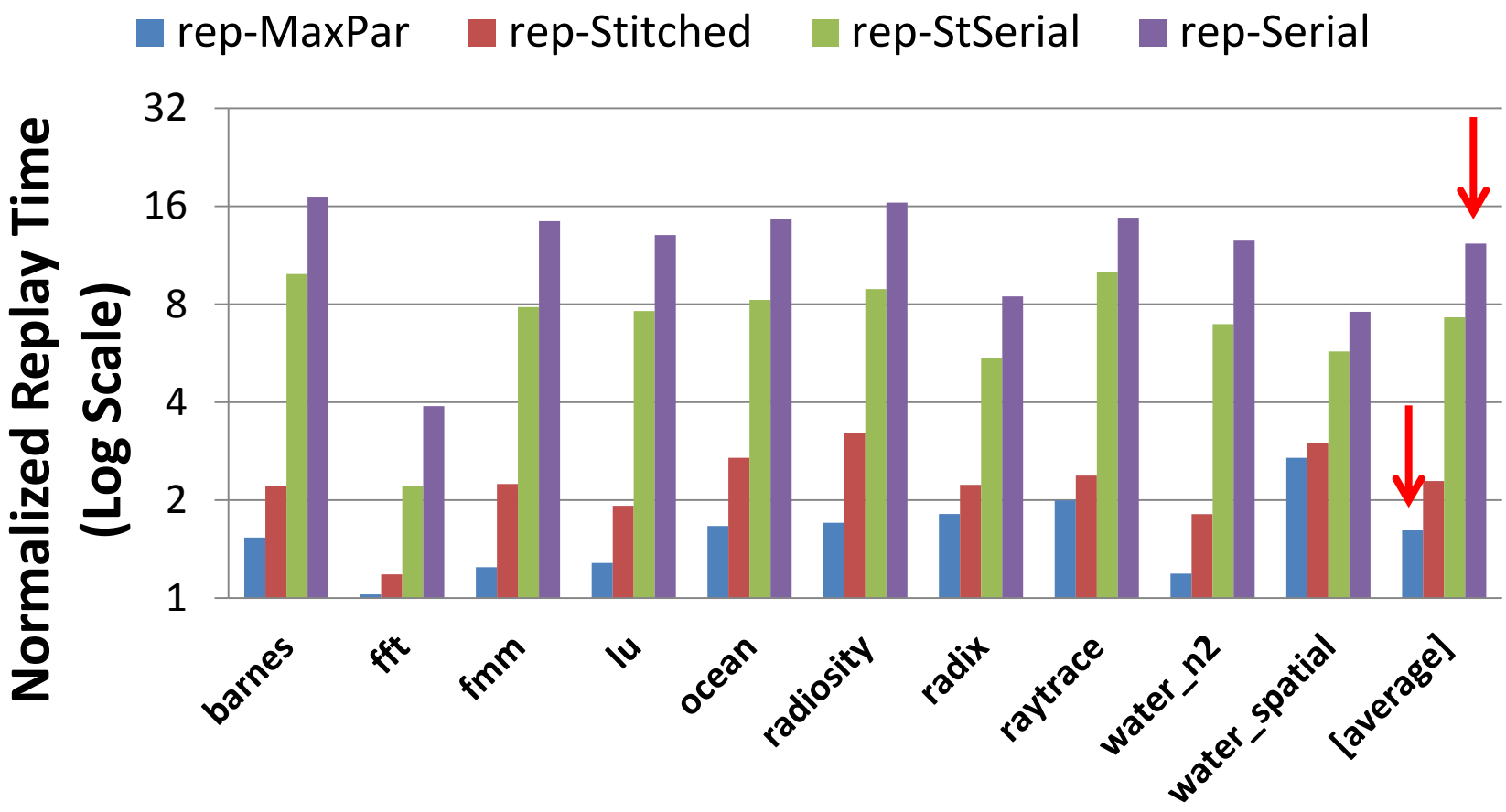



- No Parallelism
- Smallest log

# Evaluation

- Using Simics
  - Full-system simulation with OS
  - Wrote a Linux kernel module to
    - Records application inputs
    - Controls RRUs
- Model 8 + 1 processors
  - 8 processors for the app
  - 1 processor for the backend
- 10 SPLASH-2 benchmarks

# Replay Time Normalized to Recording




- Large difference between MaxPar and Serial replay
- On 8 processors, unoptimized MaxPar replay is only 50% slower than recording

# Conclusions

- Cyrus: RnR system that supports
  - Application-level RnR
  - Unintrusive hardware
  - Flexible replay parallelism
- Key idea: On-the-fly software backend pass
- On 8 processors:
  - Large difference between MaxPar and Serial replay
  - Unoptimized replay of MaxPar is only 50% slower than recording
  - Negligible recording overhead
- Upcoming ISCA'13 paper describes our FPGA RnR prototype