Dynamic Tensor Rematerialization

Presenter: Steven Lyubomirsky*

Marisa Kirisame* Altan Haan* Jennifer Brennan Mike He

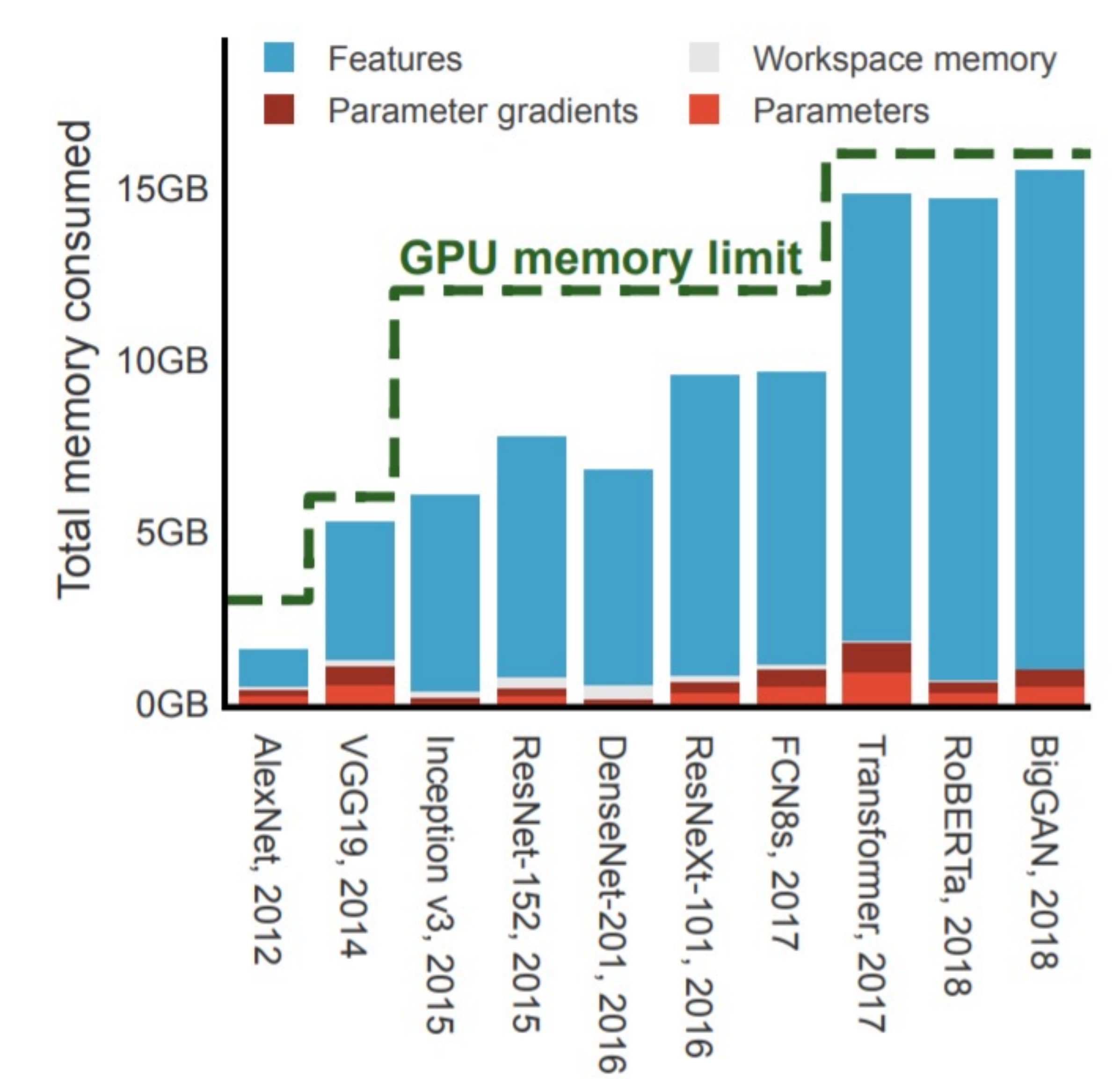
Jared Roesch Tianqi Chen Zachary Tatlock

*Equal contribution

PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

sampl

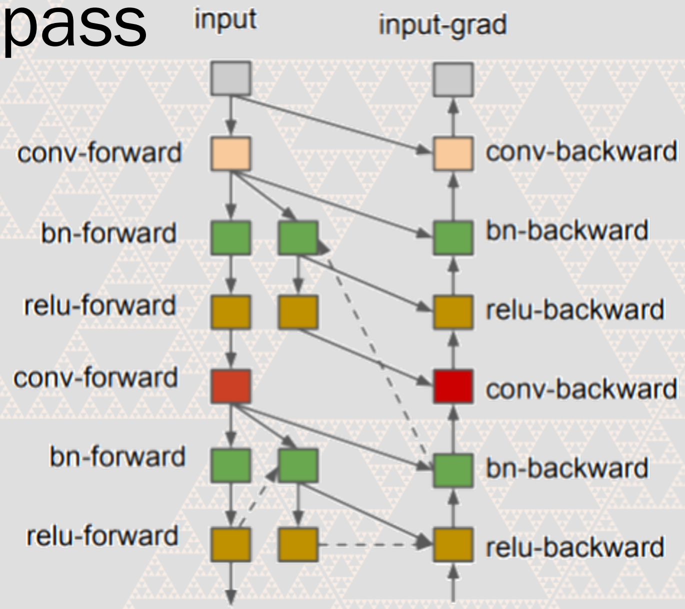
PLSE



Jain *et al.*, “Checkmate: Breaking the Memory Wall With Optimal Tensor Rematerialization” (2020)

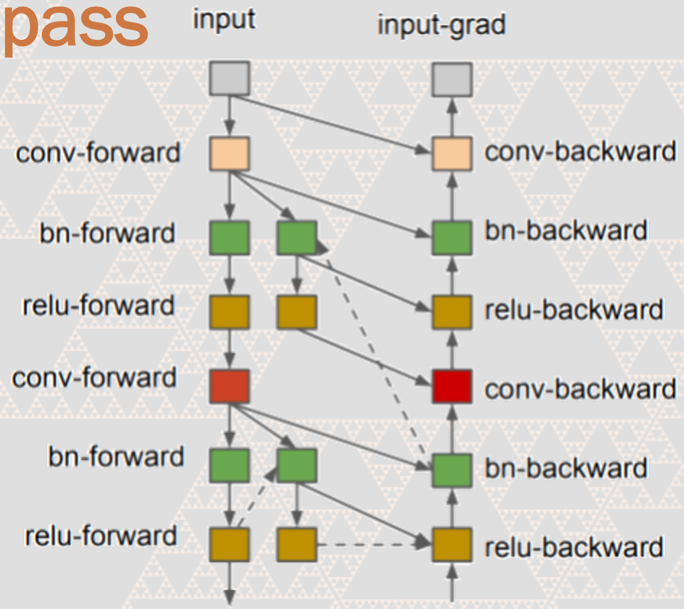
Checkpointing: Trade Time for Space

- Recompute activations instead of storing them
- Gradient Checkpointing, Chen *et al.* (2016)
  - Pick segments to recompute in backward pass
  - $O(\sqrt{N})$ memory for $O(N)$ extra ops
  - Many later segmenting approaches
- Checkmate, Jain *et al.* (2020)
  - Rematerialize individual values
  - ILP for optimal(!) planning



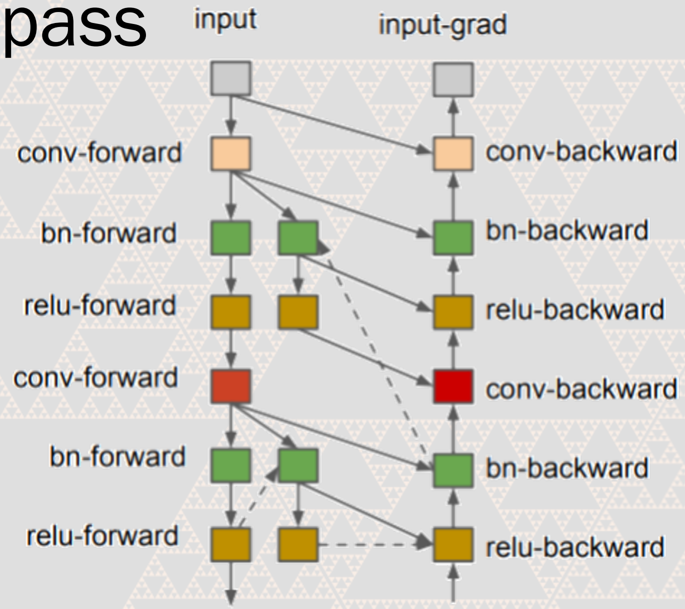
Checkpointing: Trade Time for Space

- Recompute activations instead of storing them
- Gradient Checkpointing, Chen *et al.* (2016)
  - Pick segments to recompute in backward pass
  - $O(\sqrt{N})$ memory for $O(N)$ extra ops
  - Many later segmenting approaches
- Checkmate, Jain *et al.* (2020)
  - Rematerialize individual values
  - ILP for optimal(!) planning



Checkpointing: Trade Time for Space

- Recompute activations instead of storing them
- Gradient Checkpointing, Chen *et al.* (2016)
  - Pick segments to recompute in backward pass
  - $O(\sqrt{N})$ memory for $O(N)$ extra ops
  - Many later segmenting approaches
- Checkmate, Jain *et al.* (2020)
  - Rematerialize individual values
  - ILP for optimal(!) planning



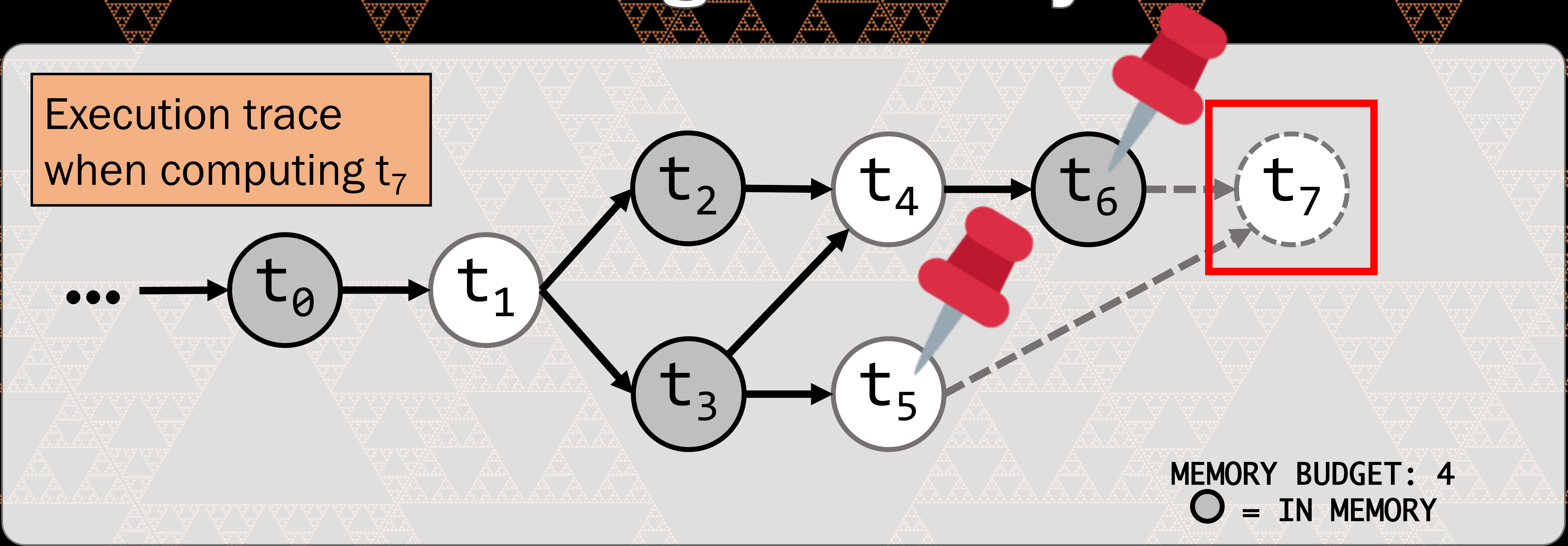
Static Planning is Unnecessary

- Past approaches plan checkpoints in advance
- Require static knowledge of the model
- Planning can be expensive, limits applications
- Our contributions:
    - *Static planning is unnecessary for checkpointing*
    - Still achieve good compute-memory tradeoffs

Dynamic Tensor Rematerialization

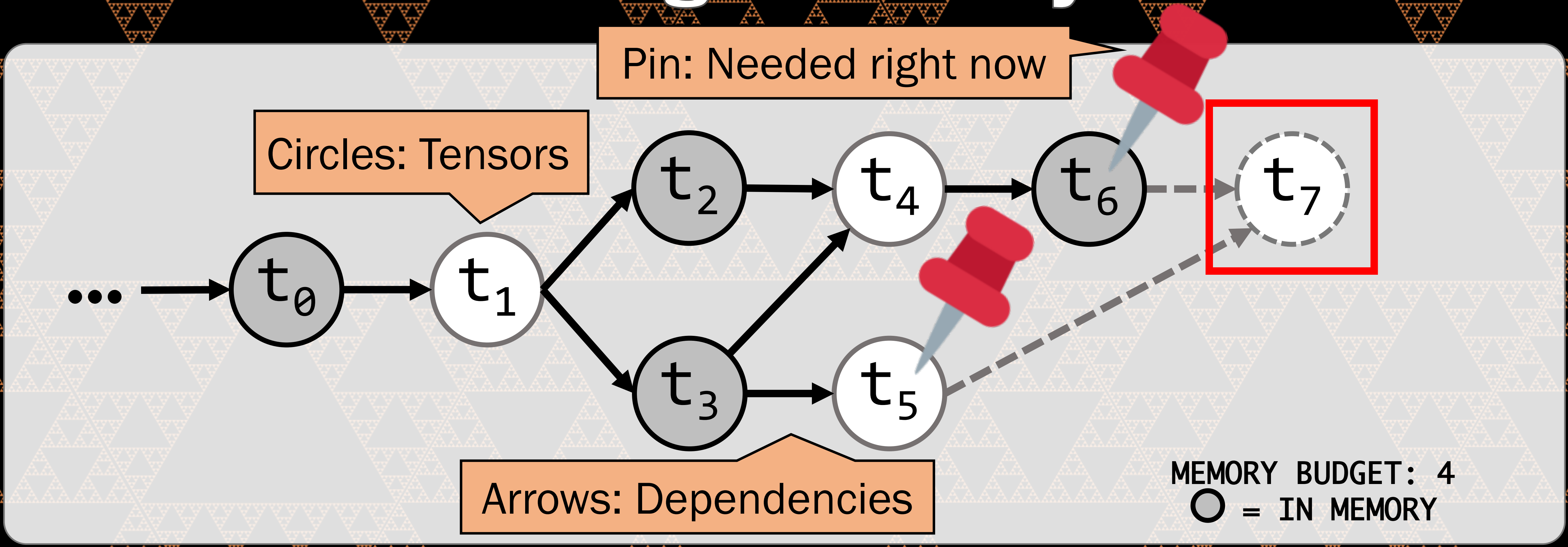
- Cache-like approach: A runtime system
  - No static information necessary
  - Greedily allocate, evict and recompute as needed
  - Collects metadata to guide heuristics
  - Operates at a high level of abstraction
- Still competitive with static planning!

Rematerializing on the Fly



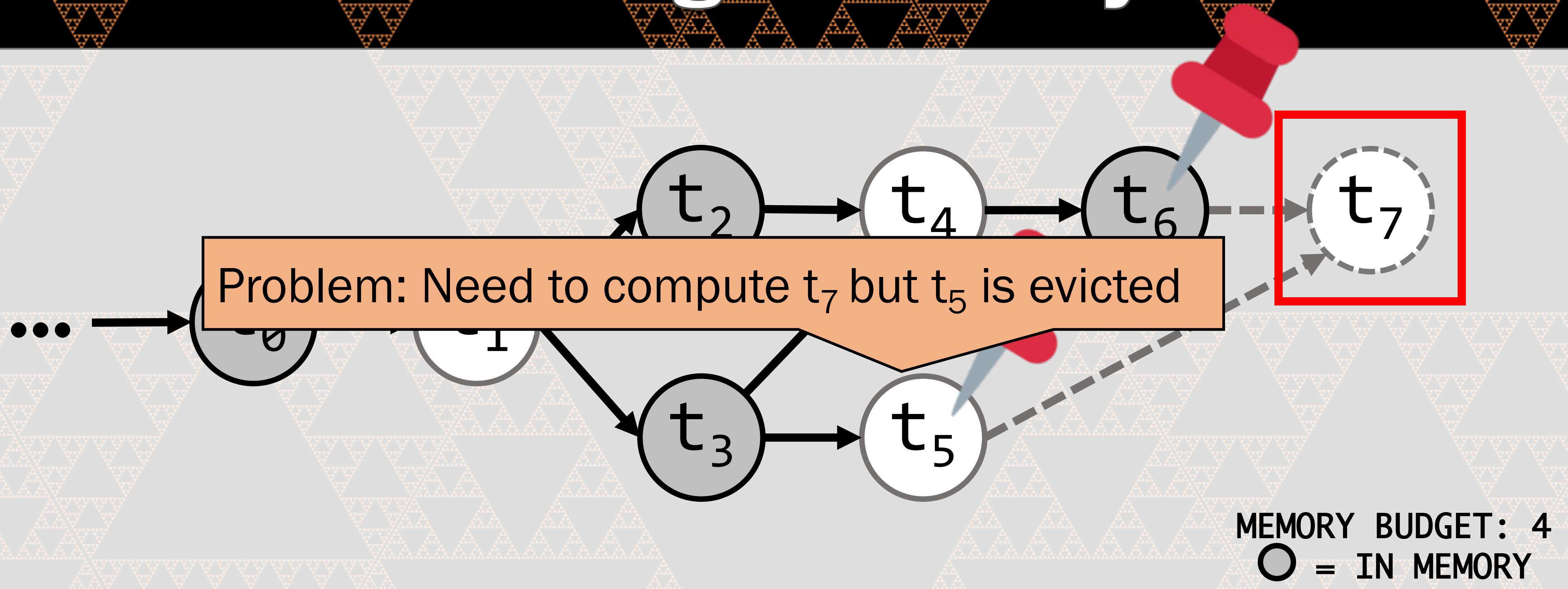
Current operation: `PerformOp(op₇, [t₅, t₆])`

Rematerializing on the Fly



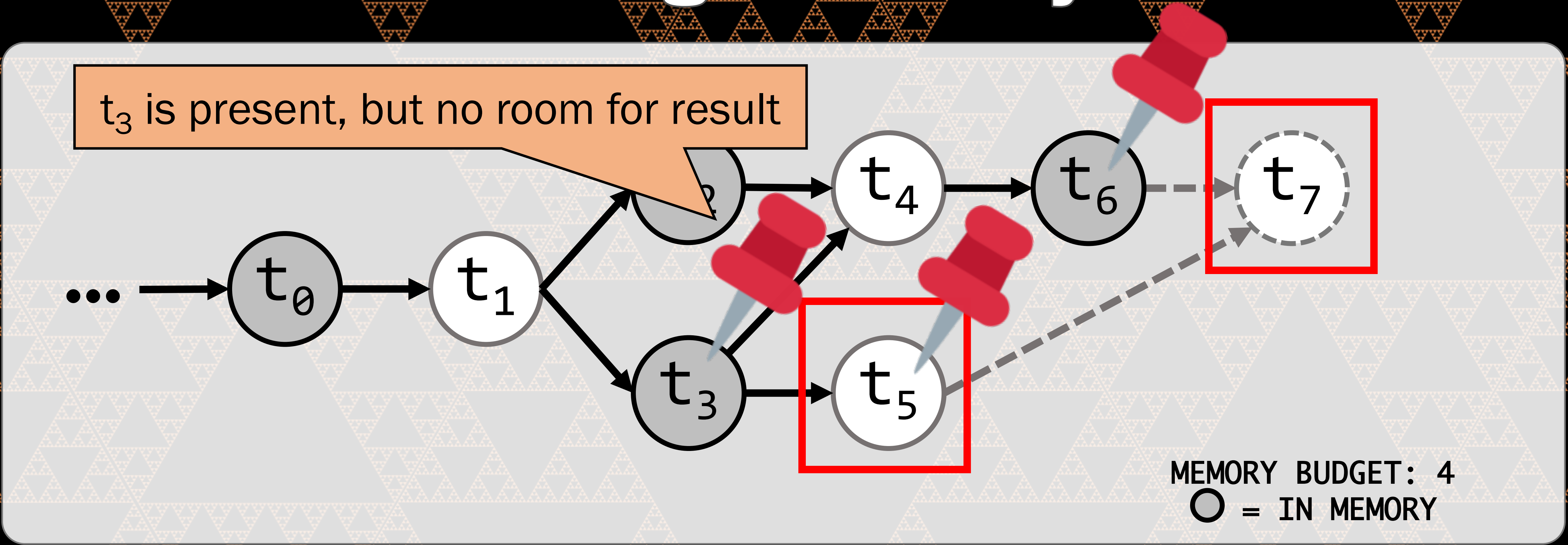
Current operation: `PerformOp(op₇, [t₅, t₆])`

Rematerializing on the Fly



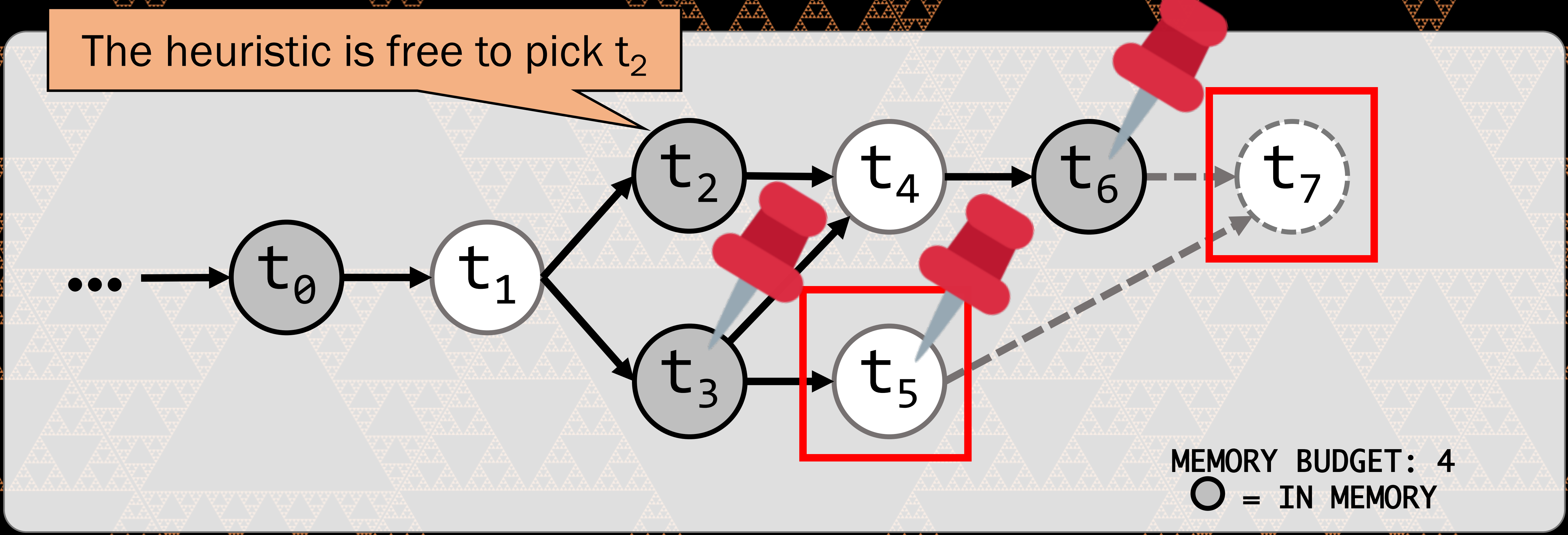
Current operation: `Rematerialize(t₅)`

Rematerializing on the Fly



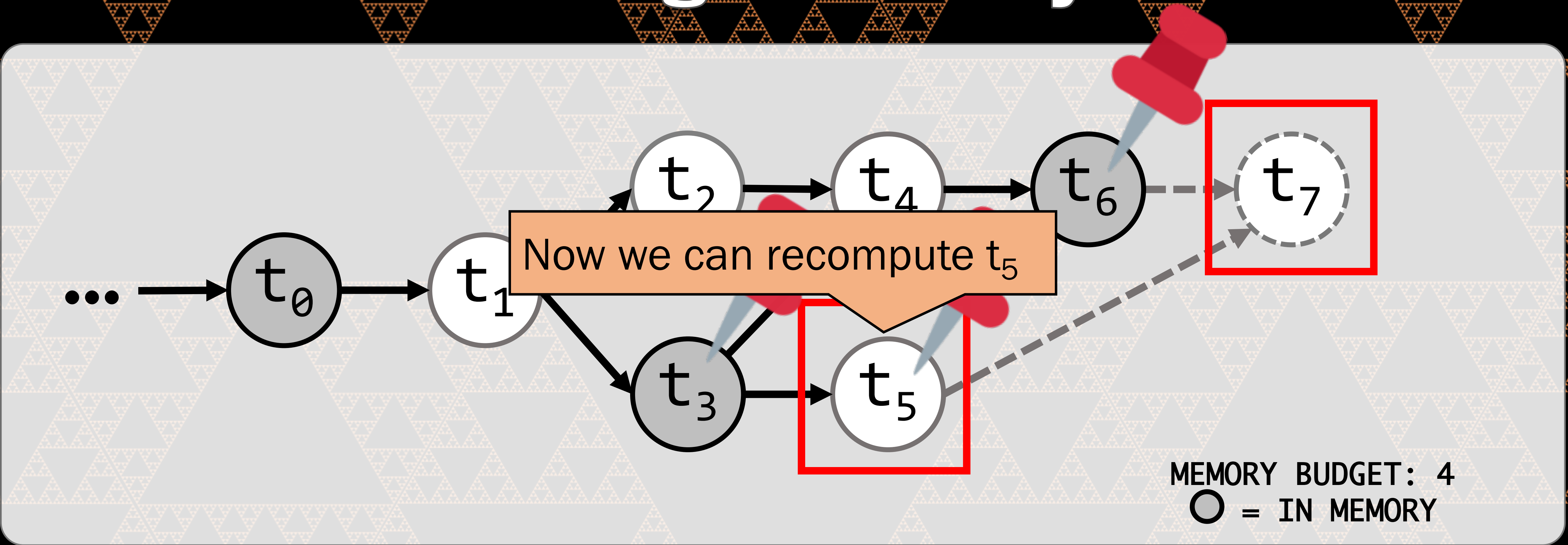
Current operation: `PerformOp(op₅, [t₃])`

Rematerializing on the Fly



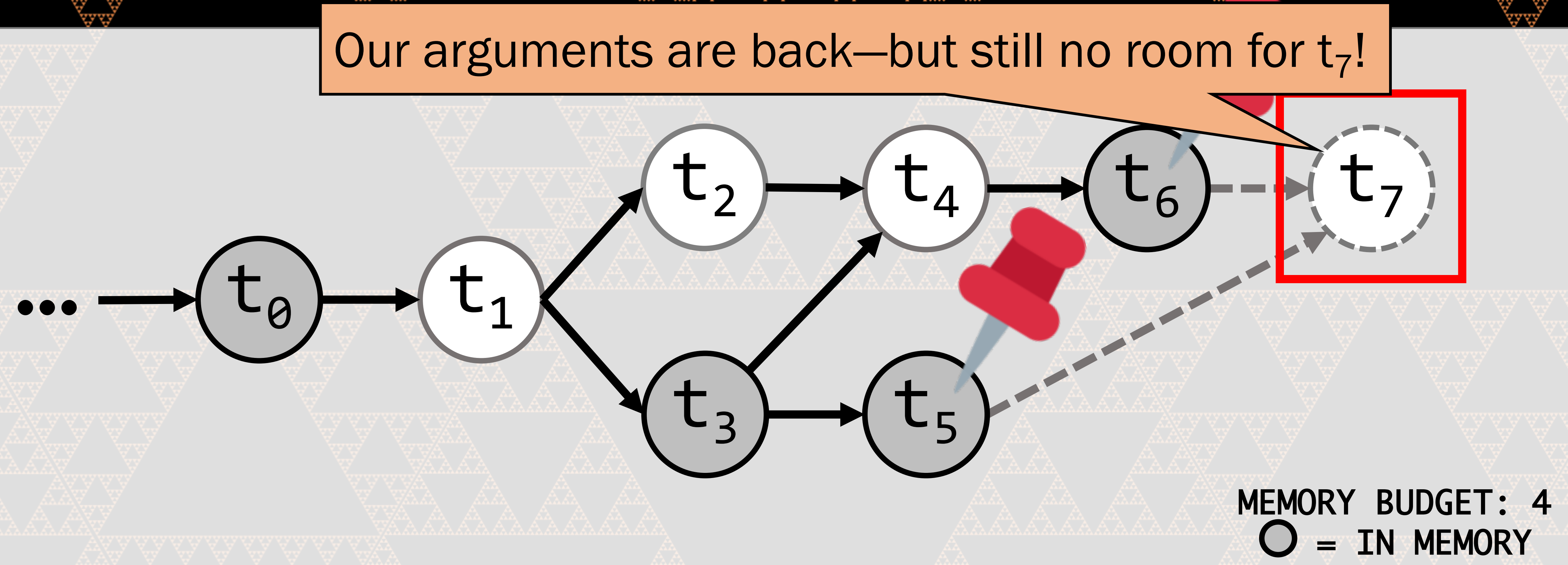
Current operation: `PerformEviction()`

Rematerializing on the Fly



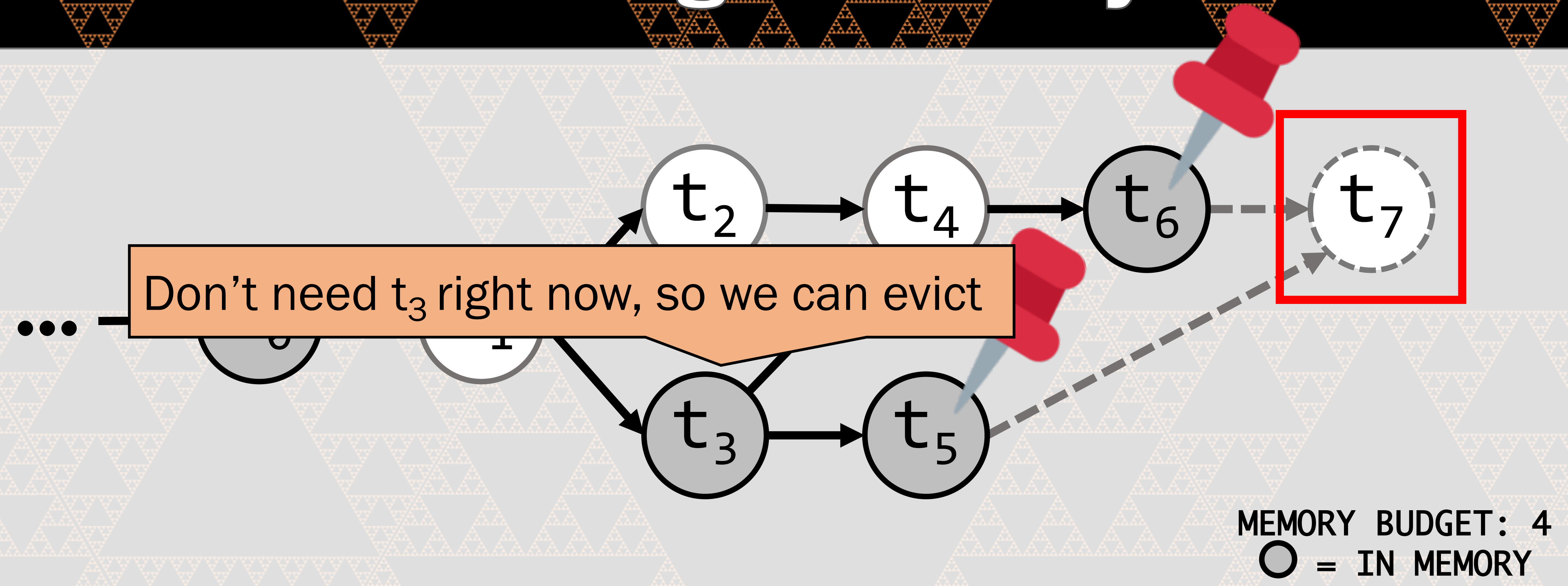
Current operation: `AllocateBuffer(t₅.size); op₅(t₃)`

Rematerializing on the Fly



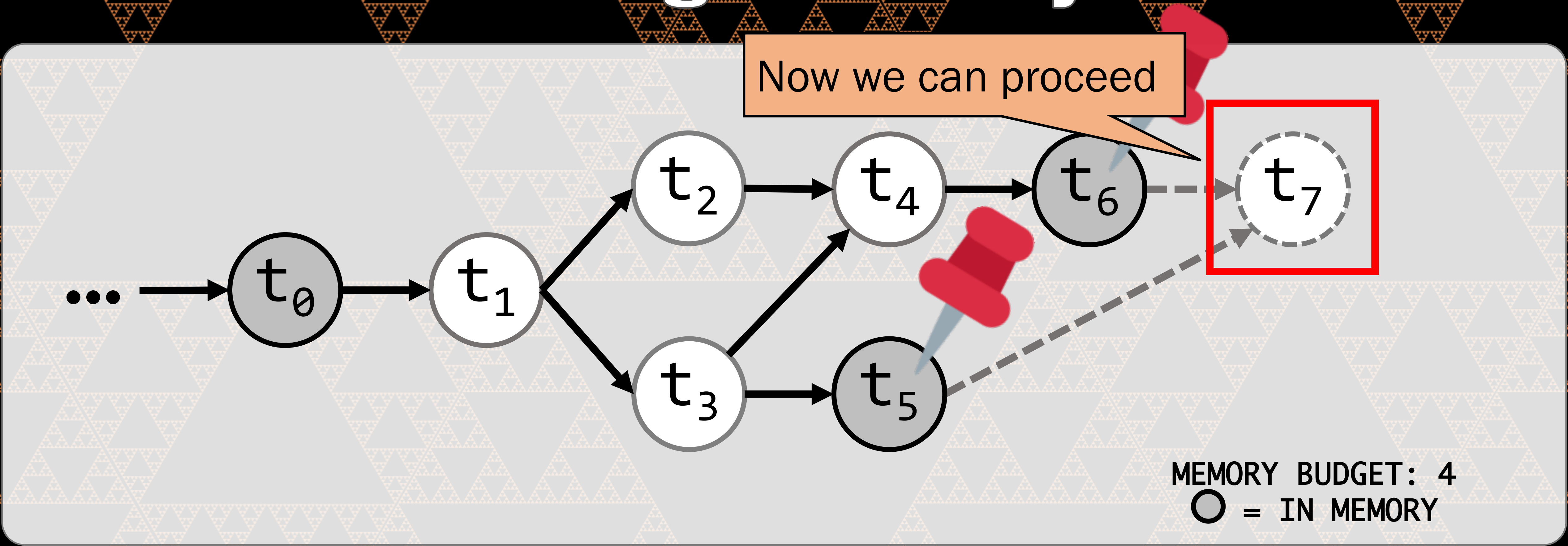
Current operation: `AllocateBuffer(t₇.size)`

Rematerializing on the Fly



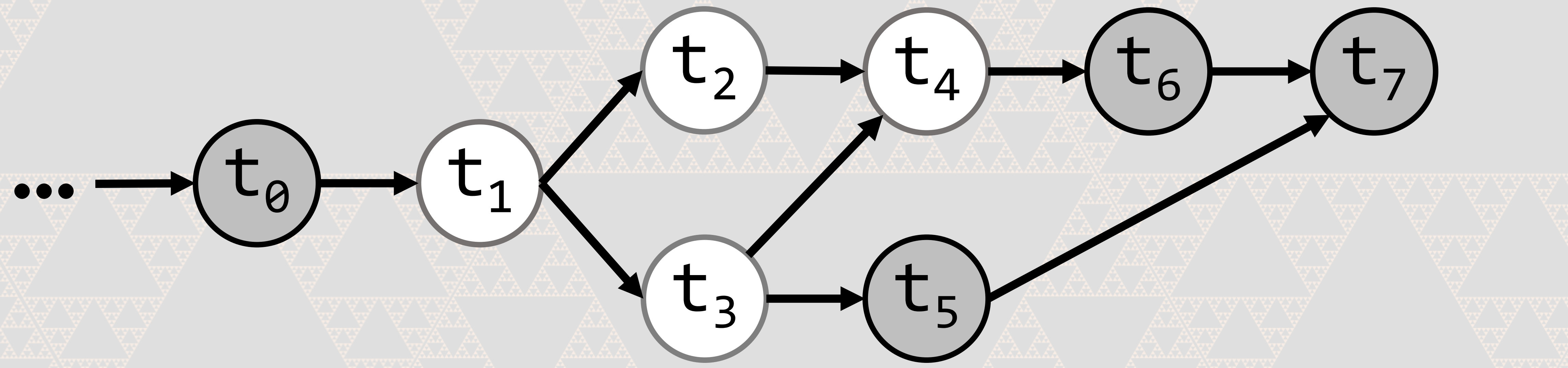
Current operation: `PerformEviction()`

Rematerializing on the Fly



Current operation: $op_7(t_5,\ t_6)$

Rematerializing on the Fly



DTR: Just Some Callbacks

`AllocateBuffer(size)`: Allocate if enough room, else evict until there is

`PerformEviction()`: Heuristic chooses a tensor to evict

`Rematerialize(t)`: Recompute `t` by replaying its parent op (`PerformOp`)

`PerformOp(op, args)`:

- Rematerialize evicted arguments
- Make room for result
- Update metadata

What Do Heuristics Look Like?

- Dynamic prediction of which tensor is least valuable
- Useful metadata, easy to track:
  - Cost $c(t)$: Avoid recomputing expensive tensors
  - Staleness $s(t)$: Recently used $\Longrightarrow$ likely to be used soon
  - Memory $m(t)$: Large tensors are most profitable to evict
- Resulting policy: minimize $h(t) = c(t)/(m(t) \cdot s(t))$
- Others: LRU $\left(\frac{1}{s(t)}\right)$ and largest-first $\left(\frac{1}{m(t)}\right)$

Reasoning About Tensor Cost

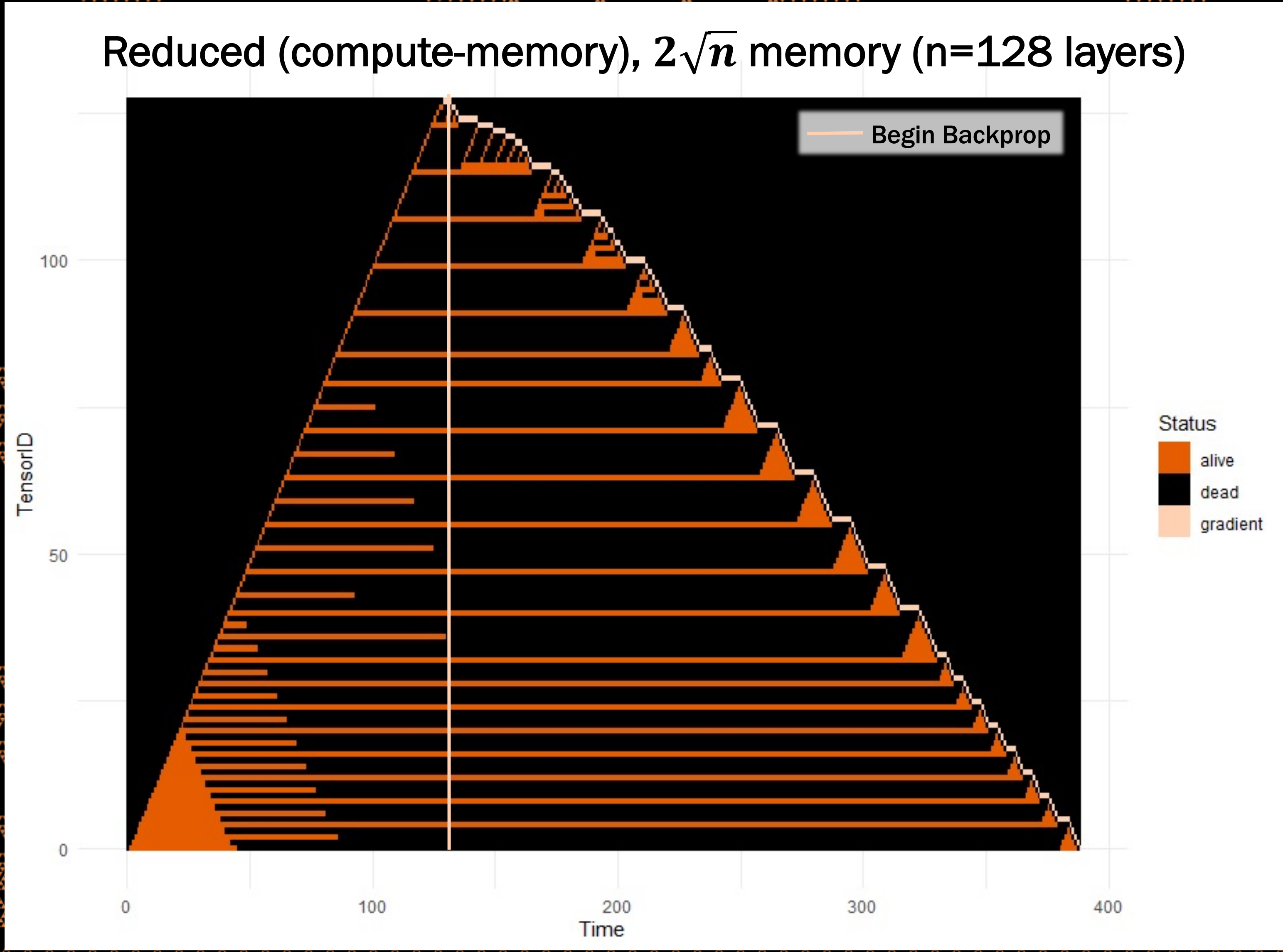
- True cost of a rematerialization includes recursive calls
- Recursively computing exact cost is expensive!
- We approximate evicted components via union-find
  - Keep a running sum for union-find components
  - When tensor rematerialized, map to a new component
  - Leaves “phantom connections” but is fast

Formal Bounds

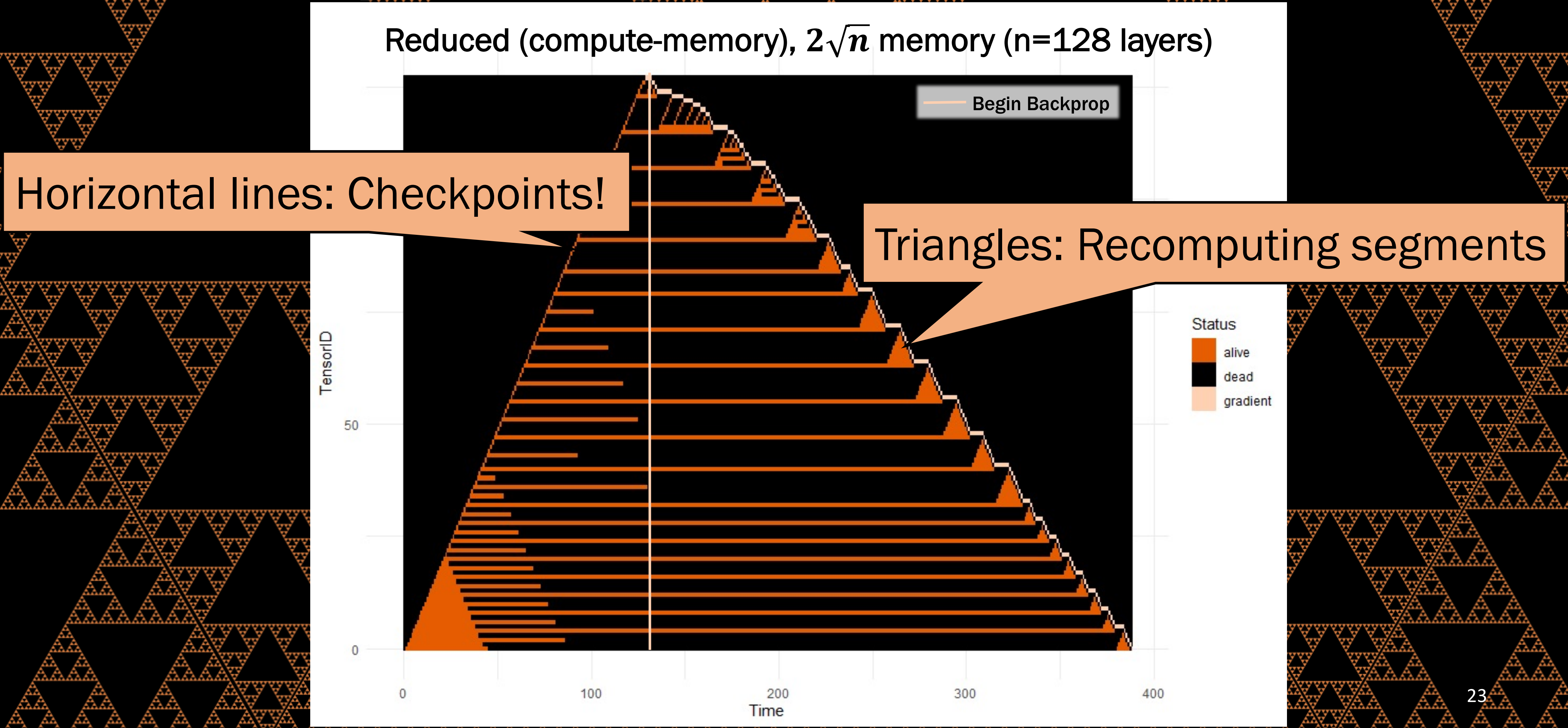
Performance on N-layer linear feedforward network:

- $\Omega(\sqrt{N})$ memory and $O(N)$ operations
- Same bound as Chen *et al*. (2016)
- No advance knowledge of model!

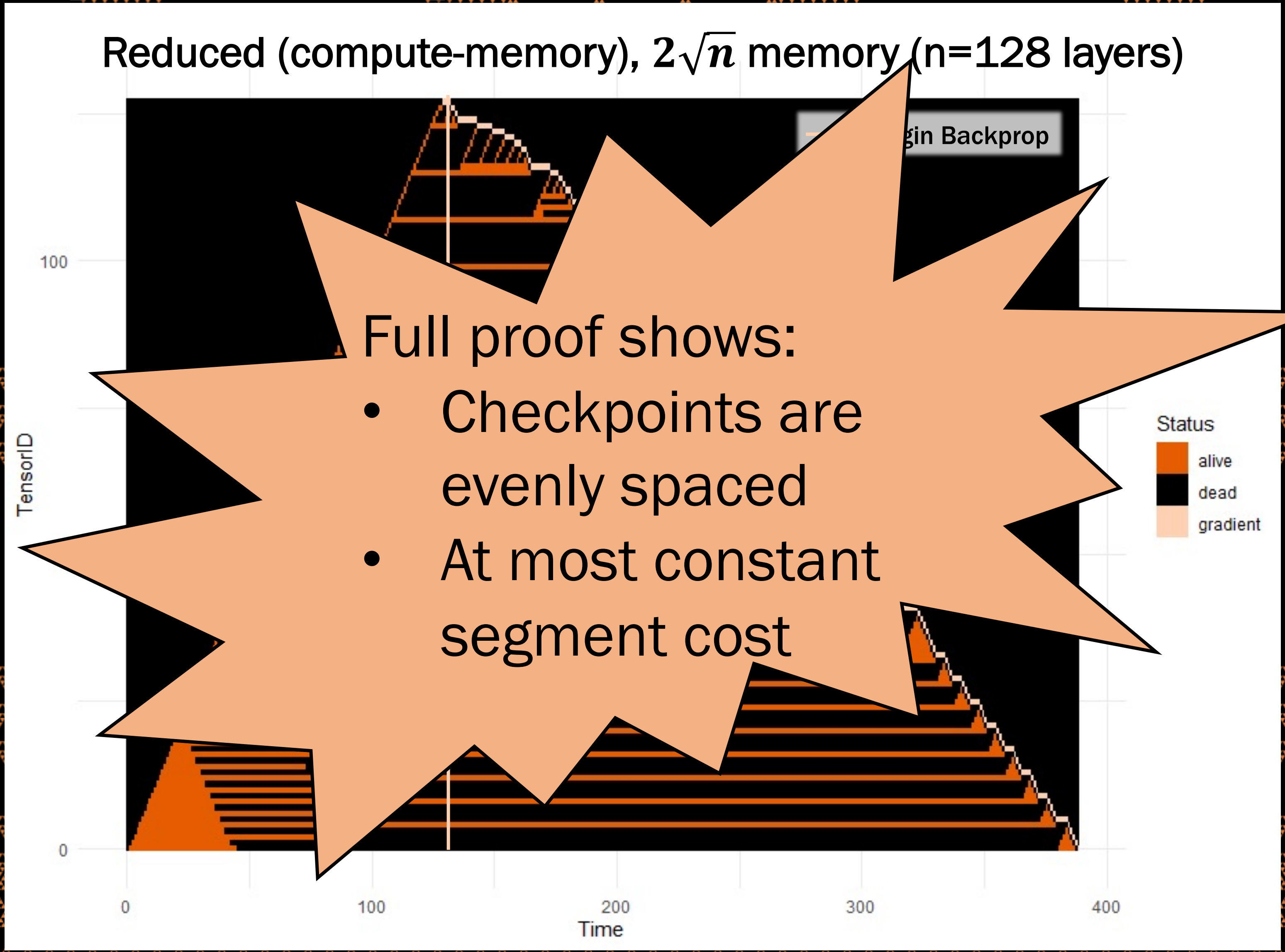
Proof (Sketch) in Pictures



Proof (Sketch) in Pictures



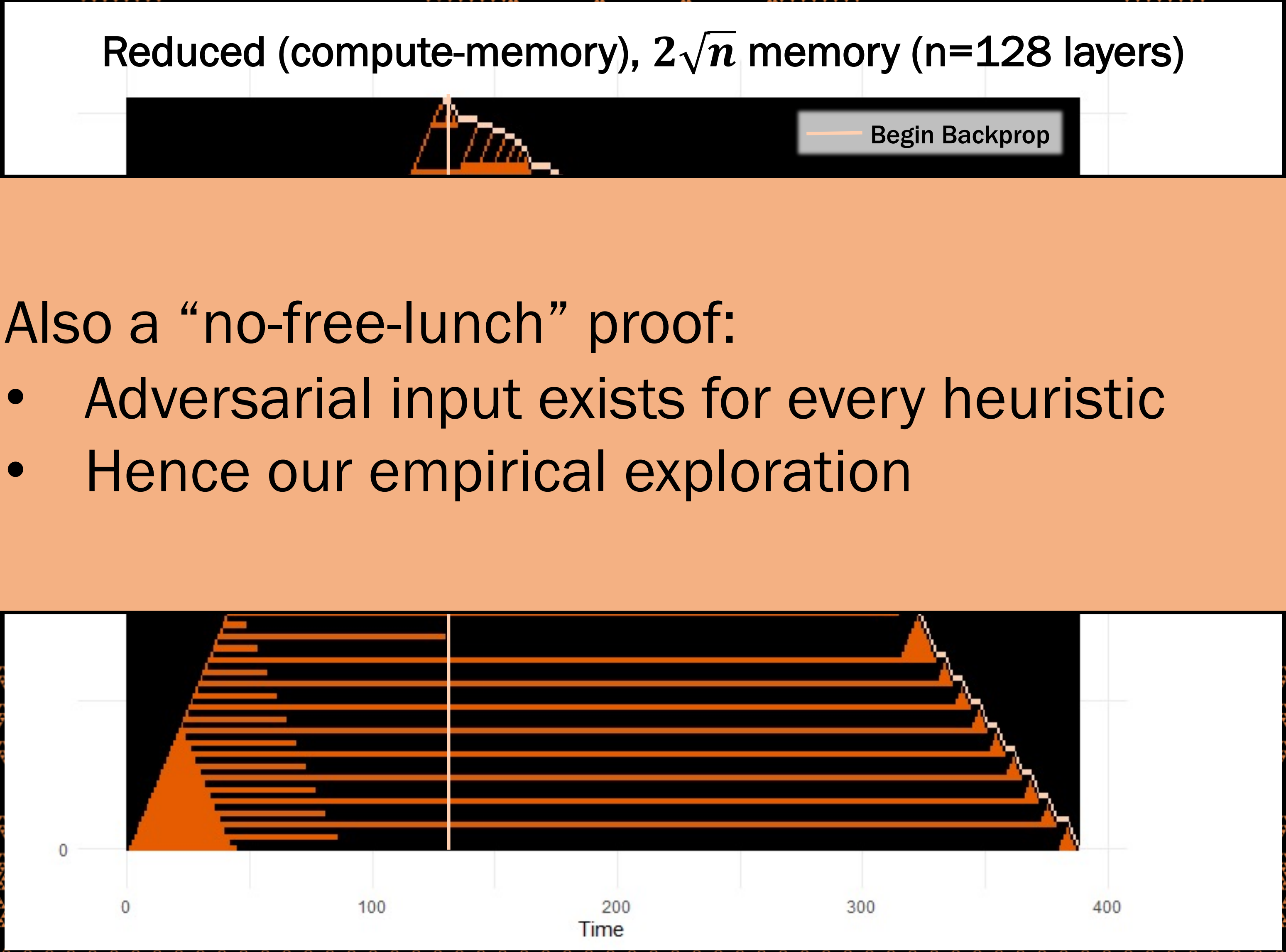
Proof (Sketch) in Pictures



Full proof shows:

- Checkpoints are evenly spaced
- At most constant segment cost

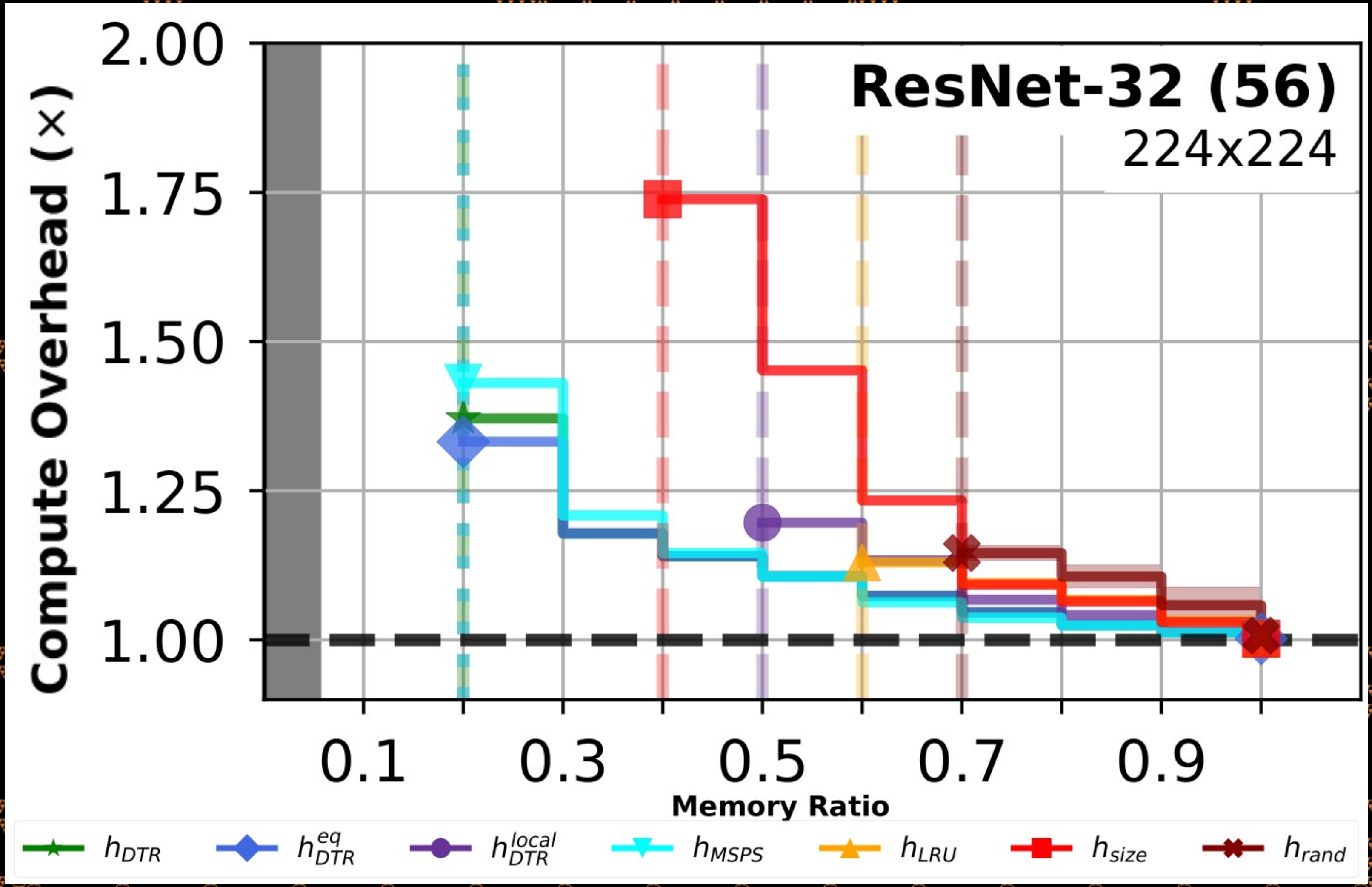
Proof (Sketch) in Pictures



Also a “no-free-lunch” proof:

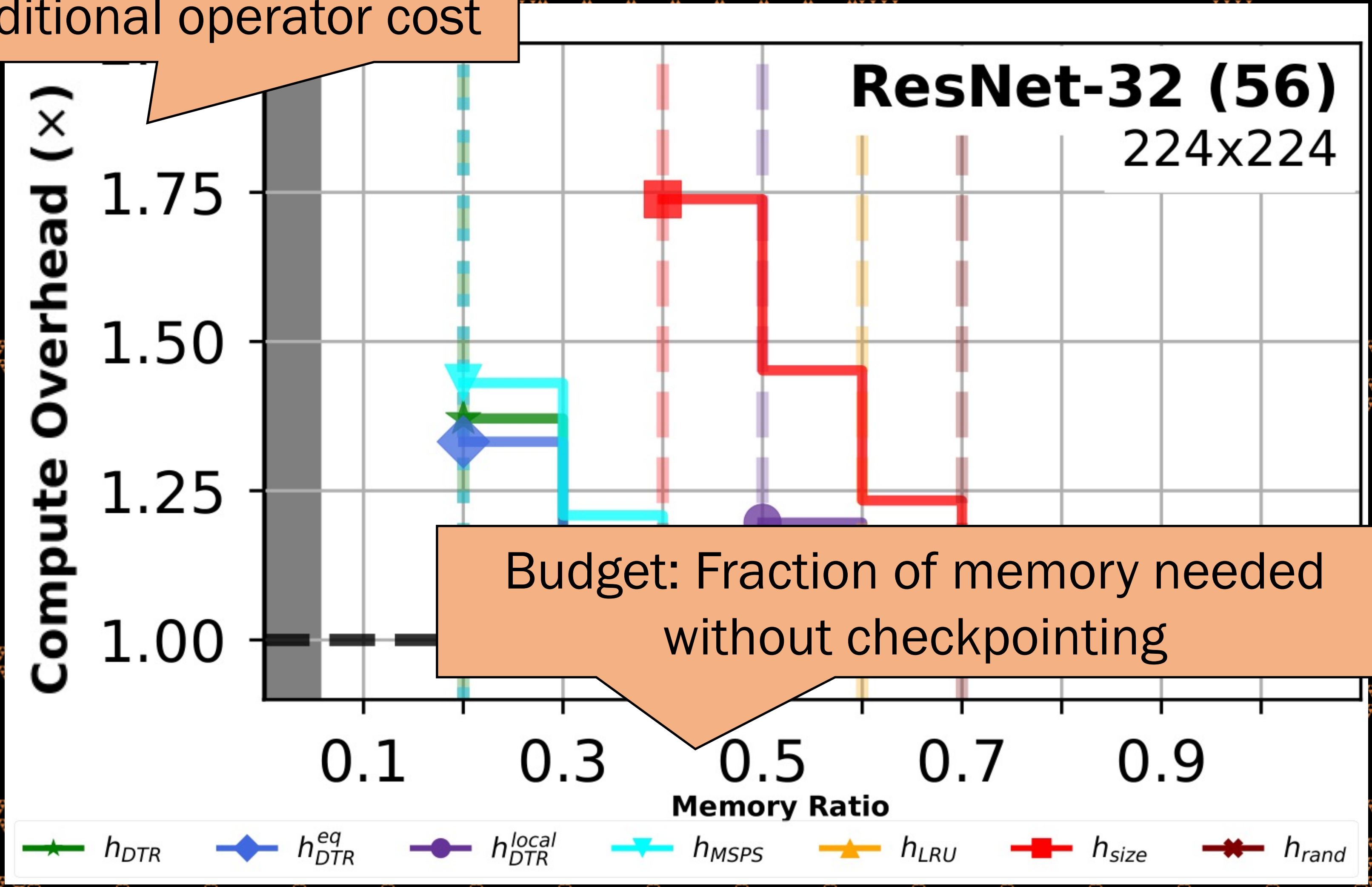
- Adversarial input exists for every heuristic
- Hence our empirical exploration

Comparison of Heuristics (Simulated)



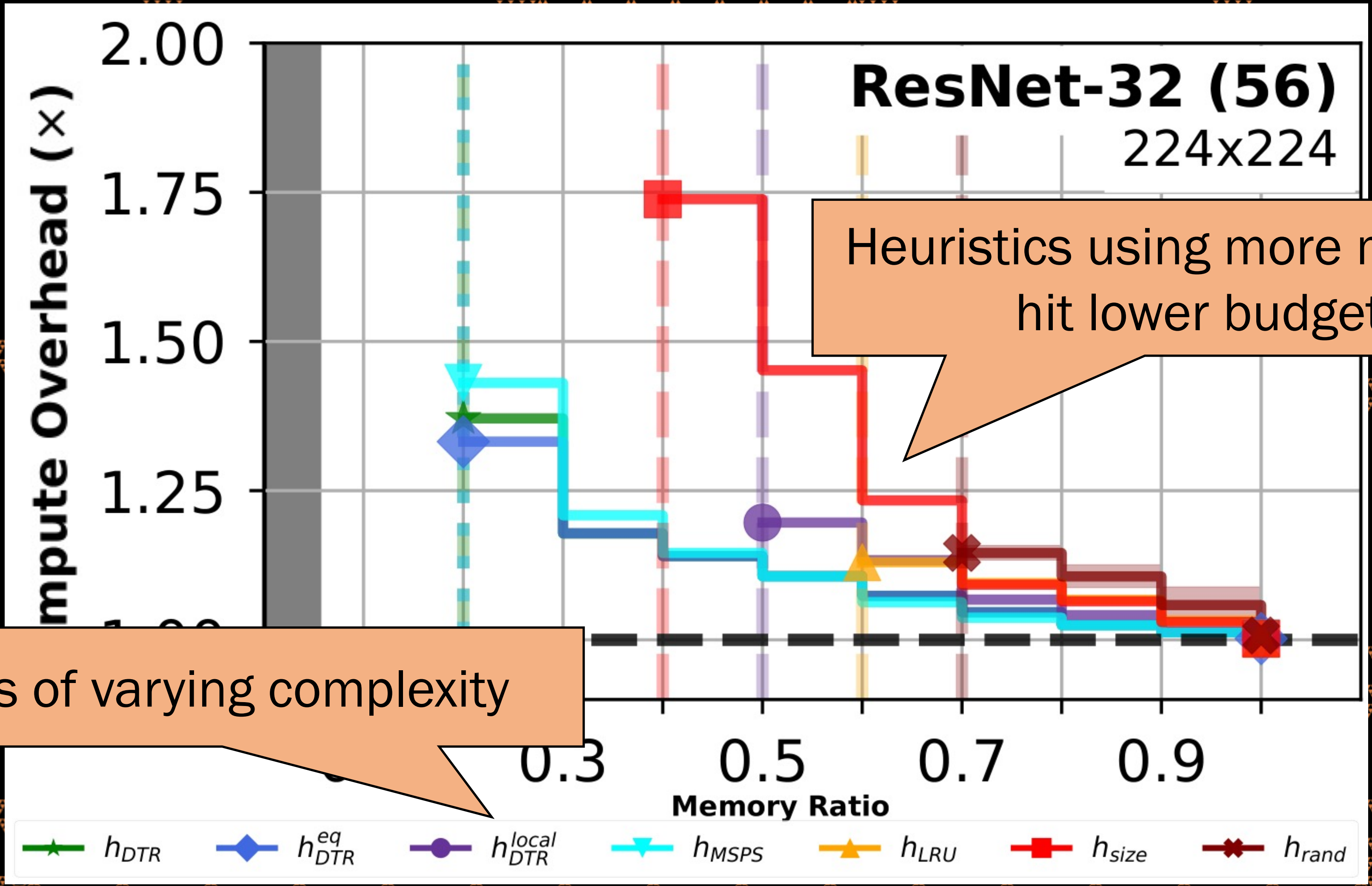
Comparison of Heuristics (Simulated)

Overhead: Additional operator cost



Budget: Fraction of memory needed without checkpointing

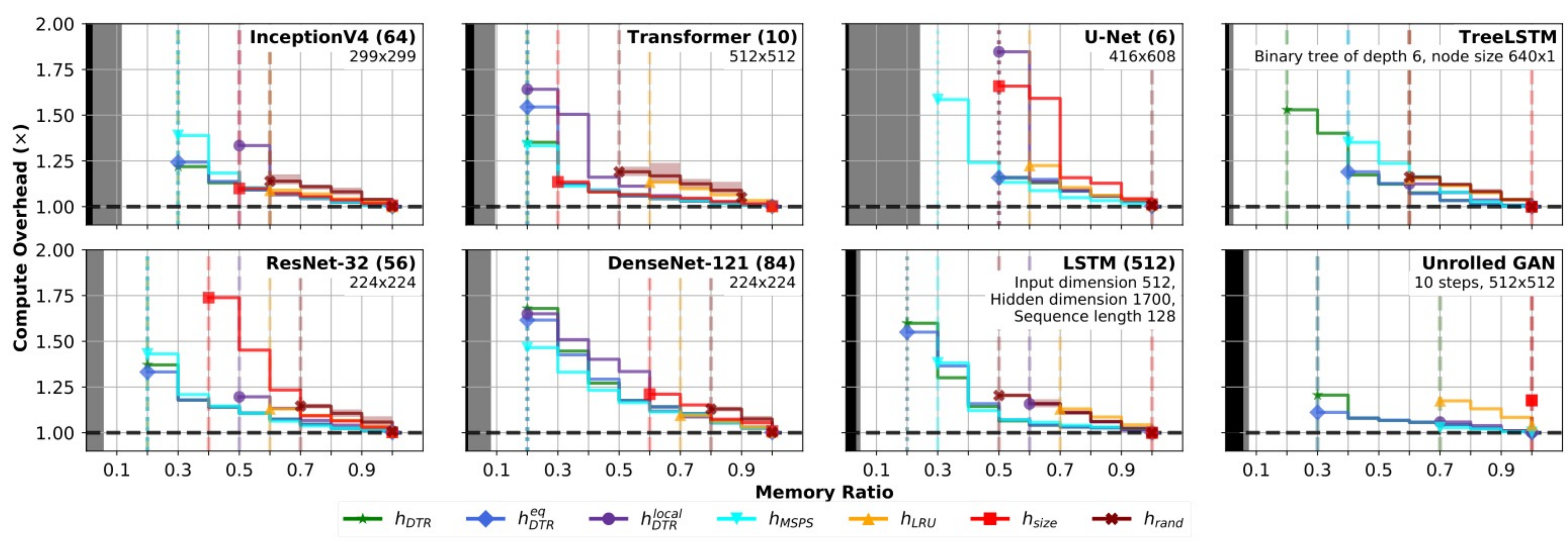
Comparison of Heuristics (Simulated)



Heuristics using more metadata hit lower budgets

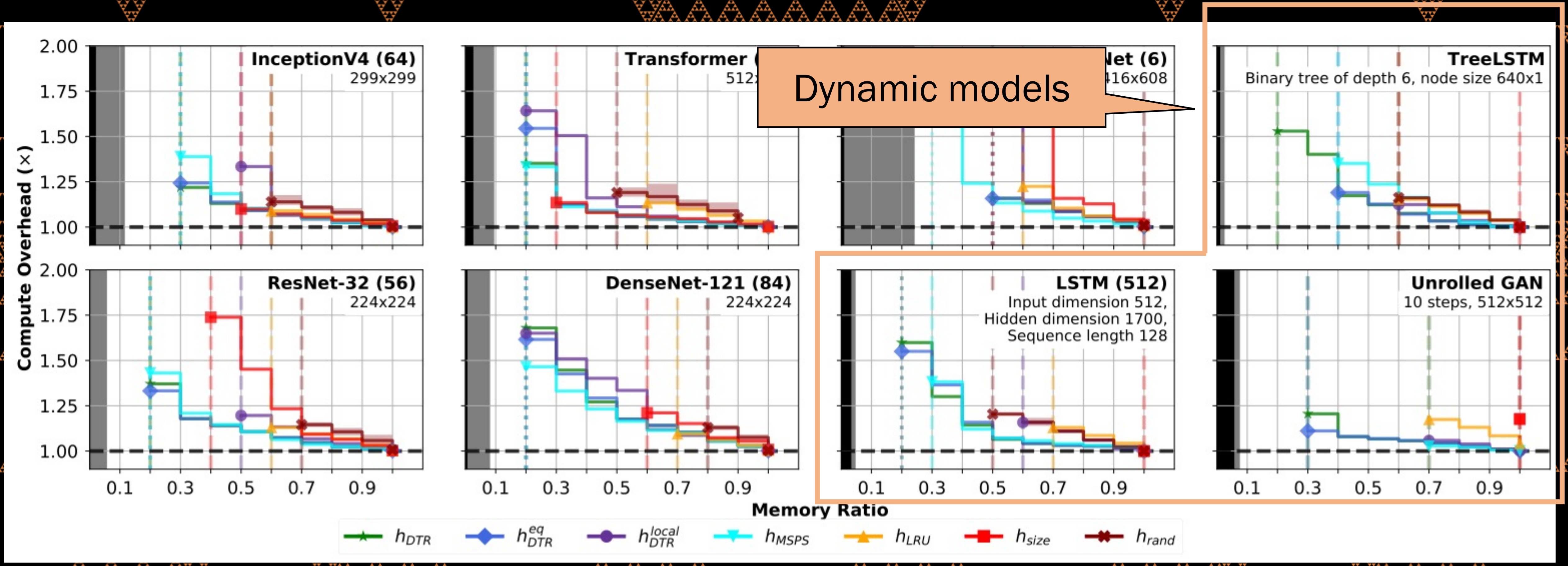
Heuristics of varying complexity

Comparison of Heuristics (Simulated)



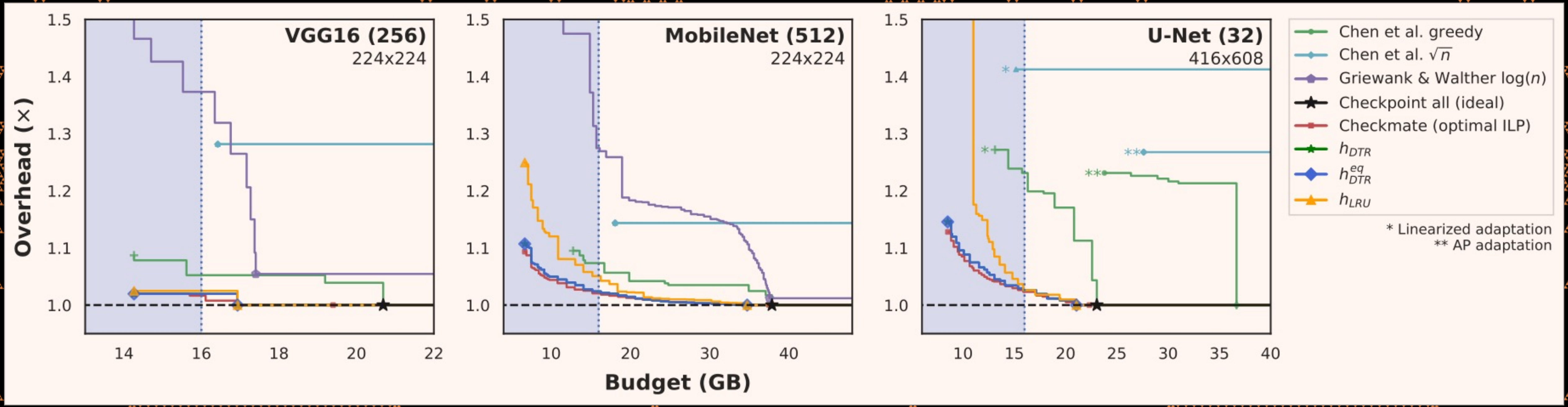
Similar trend holds across all models examined!

Comparison of Heuristics (Simulated)



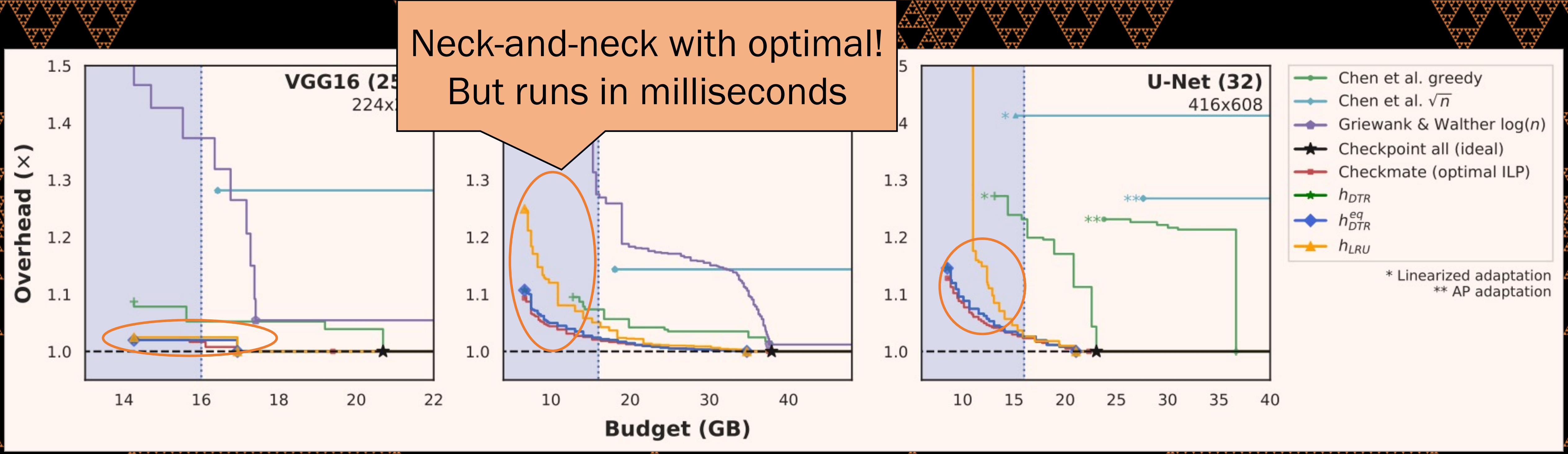
Similar trend holds across all models examined!

Comparison Against Static Techniques



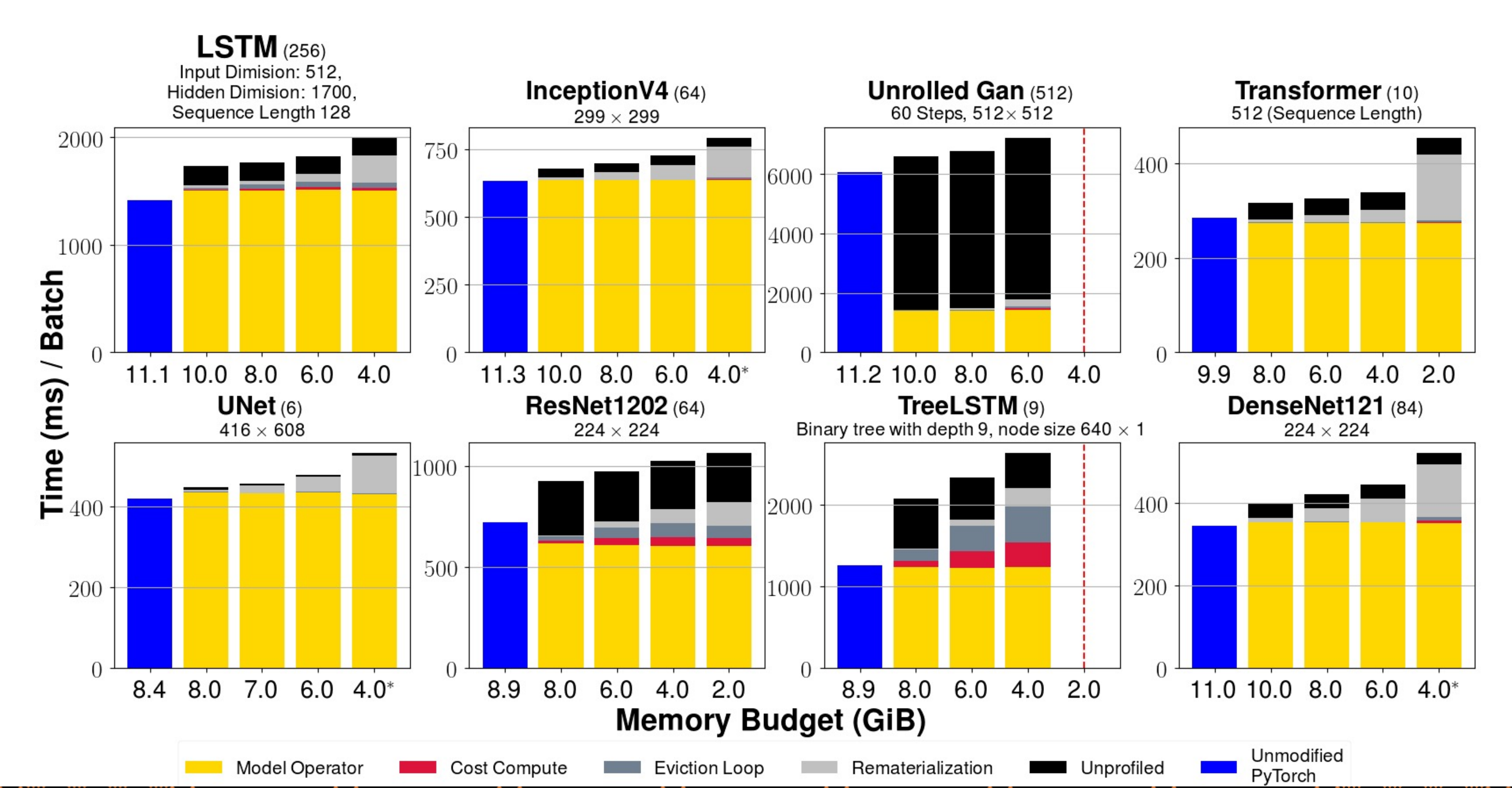
Simulated comparison via the Checkmate MLSys 2020 artifact

Comparison Against Static Techniques



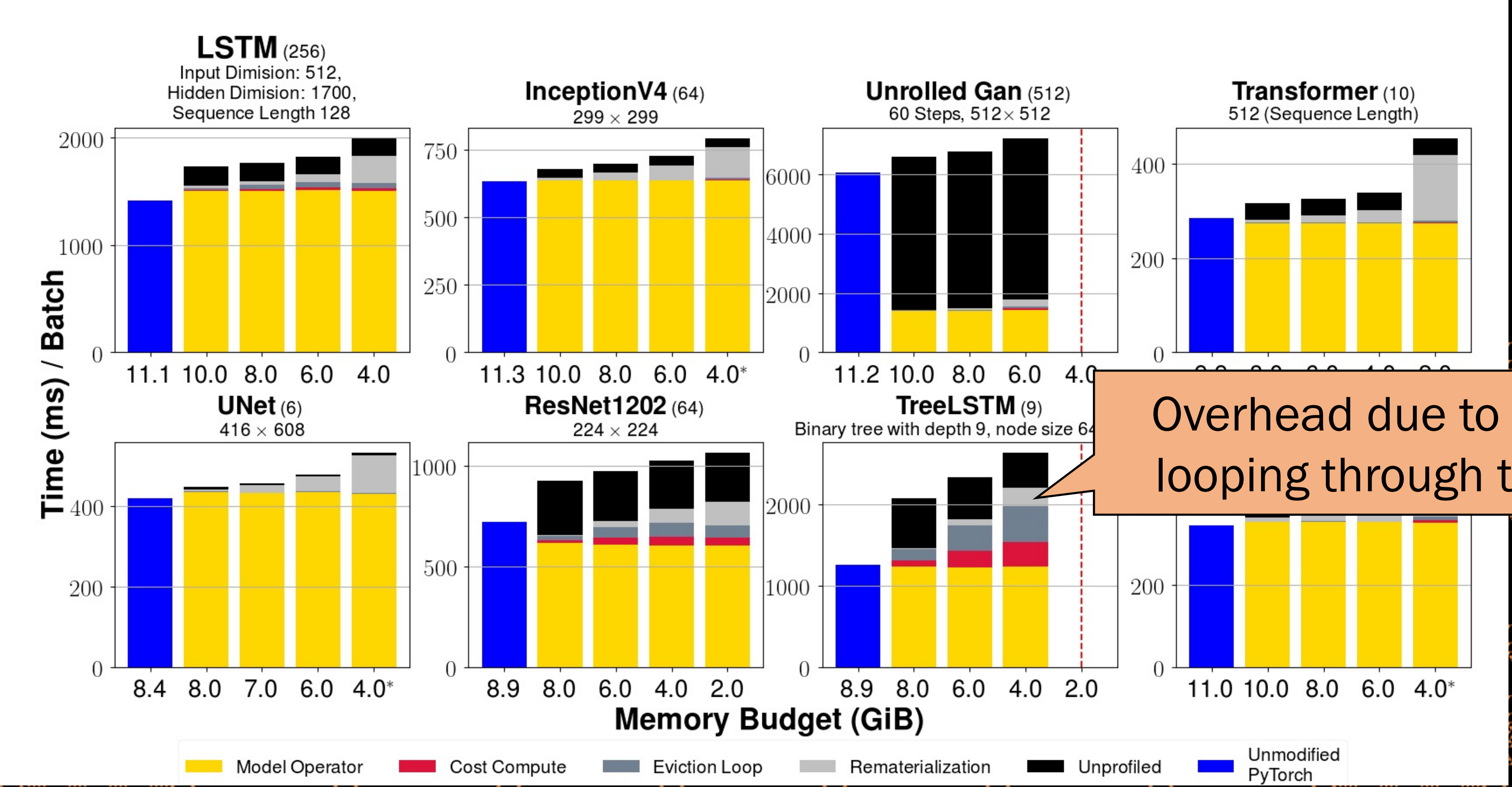
Simulated comparison via the Checkmate MLSys 2020 artifact

Prototype Implementation in PyTorch



Thin wrapper over tensor operators, core logic a few hundred LOC

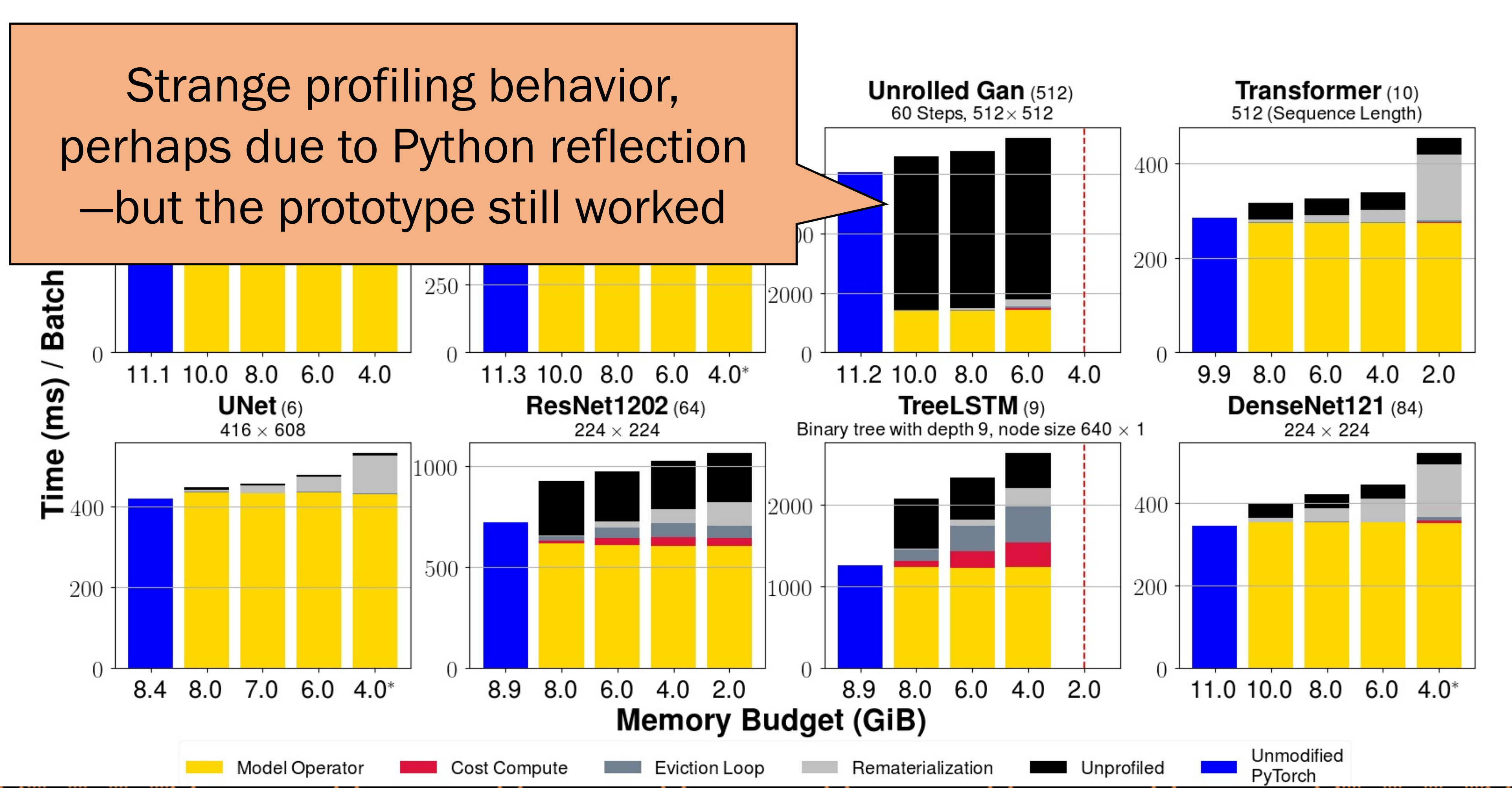
Prototype Implementation in PyTorch



Overhead due to naively looping through tensors

Thin wrapper over tensor operators, core logic a few hundred LOC

Prototype Implementation in PyTorch



Thin wrapper over tensor operators, core logic a few hundred LOC

Conclusion

- Encouraging initial results
- Many possible avenues of future work
  - Distributed settings: DTR per GPU?
  - Combining DTR with swapping
  - Tighter integration into the memory manager
  - Learning heuristics, learn from past batches
- Check out the simulator and prototype! https://github.com/uwsampl/dtr-prototype

Marisa Kirisame

Altan Haan

Jennifer Brennan

Mike He

Jared Roesch

Tianqi Chen

Zach Tatlock

JUMP

Joint University Microelectronics Program

www.src.org/program/jump

Semiconductor Research Corporation

@srcJUMP

Applications Driving Architectures