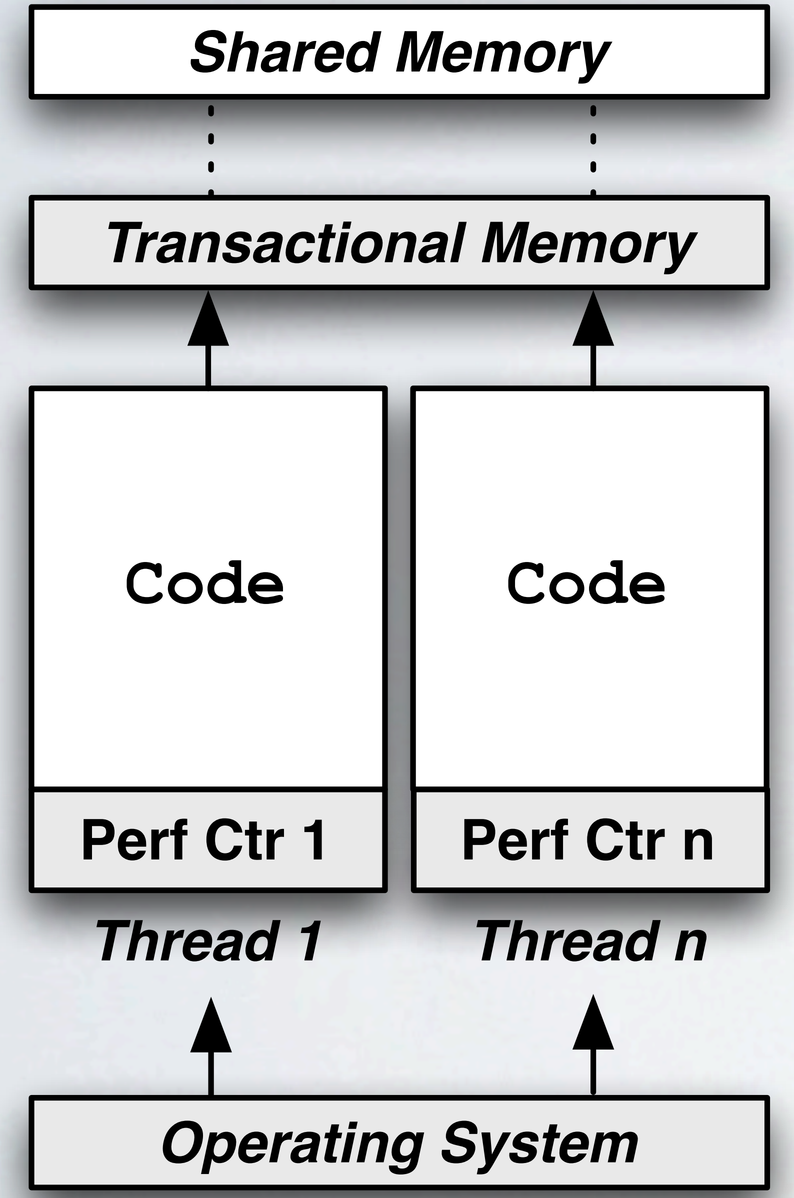
THE UNIVERSAL TRANSACTIONAL MEMORY CONSTRUCTION

Jons-Tobias Wamhoff and Christof Fetzer
Dresden University of Technology, Germany

MOTIVATION

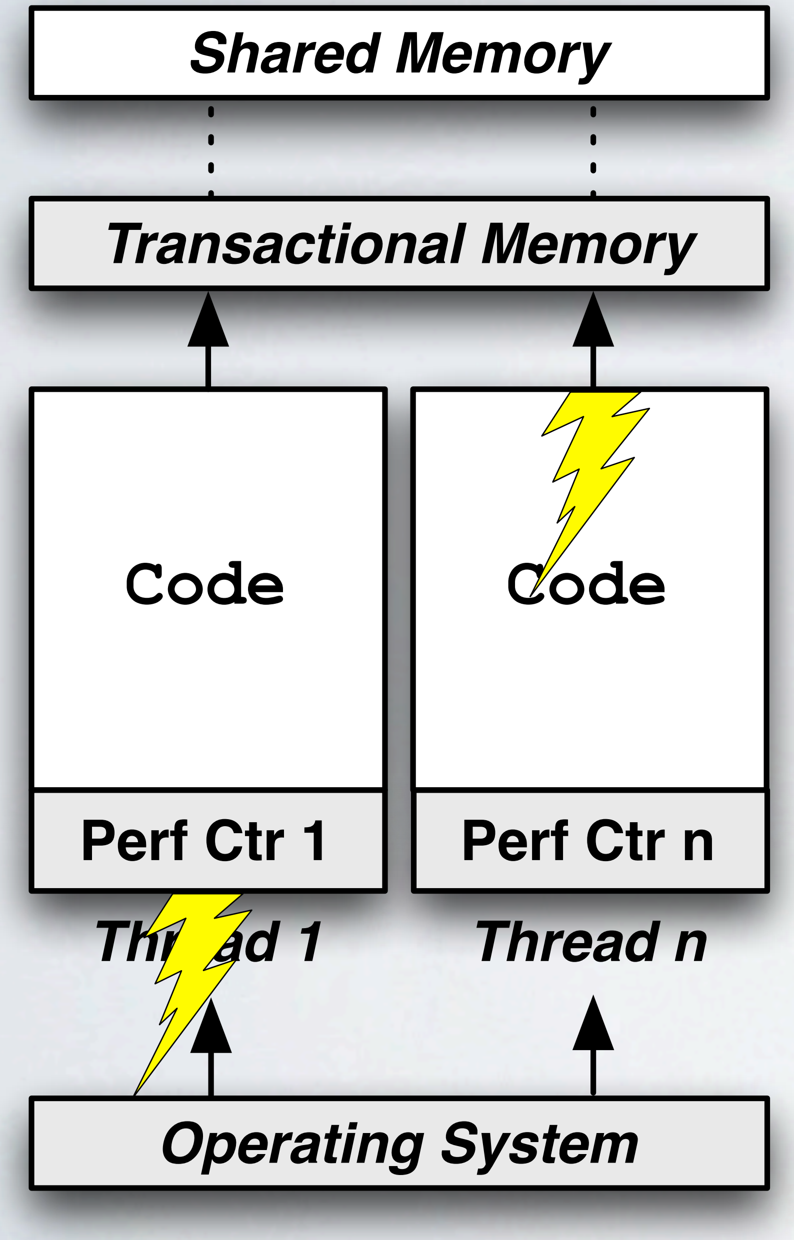
- Universal construction
  - Shows how to convert sequential algorithm into concurrent wait-free algorithm
  - Can one base such a construction on TM?
    - Wait-free progress for all correct operations
    - Tolerate crashes and non-terminating operations

ASYNCHRONOUS MULTICORE SYSTEM MODEL (AMSM)



- *Asynchronous* model with features of current multi-core systems
  - *Performance counters* for executed cycles per thread
  - Size of *memory is bound*
  - Operations (transactions) can be executed by any thread
  - *Compare-and-Swap* (CAS), *Fetch-and-Increment* (FAI)

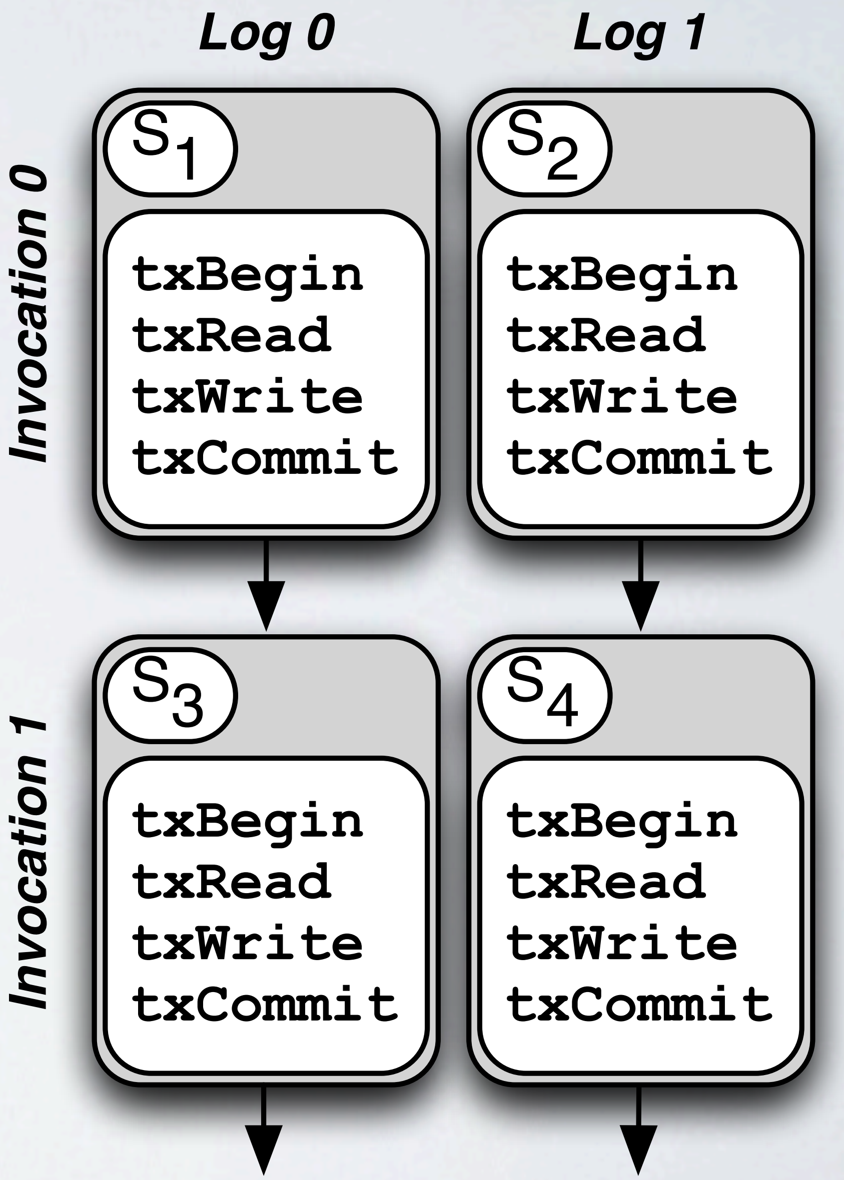
AMSM CRASH FAILURES



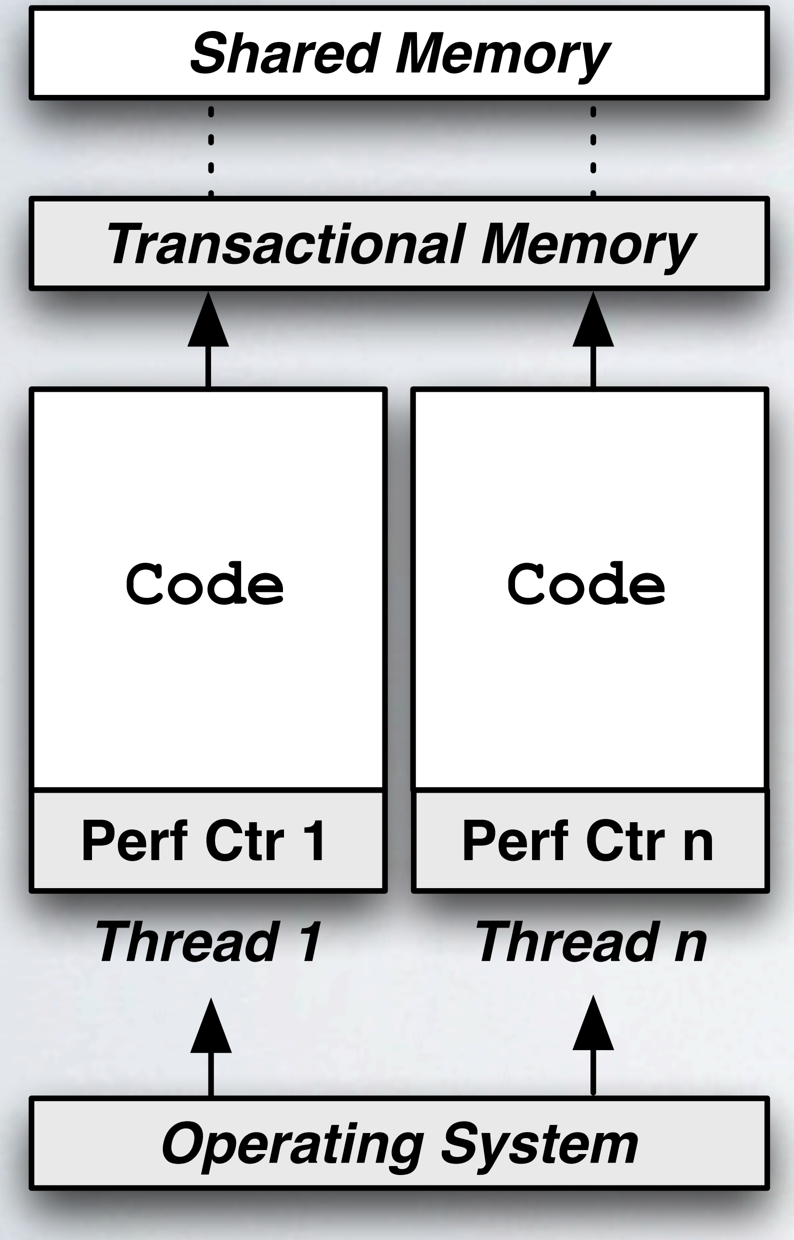
- Threads can *crash* (stop taking steps) caused by:
  - Operating system, hardware, signal: *not detectable* in AMSM
  - Program code (bug): detected by runtime and converted in *exception*

AMSM PROGRAMS

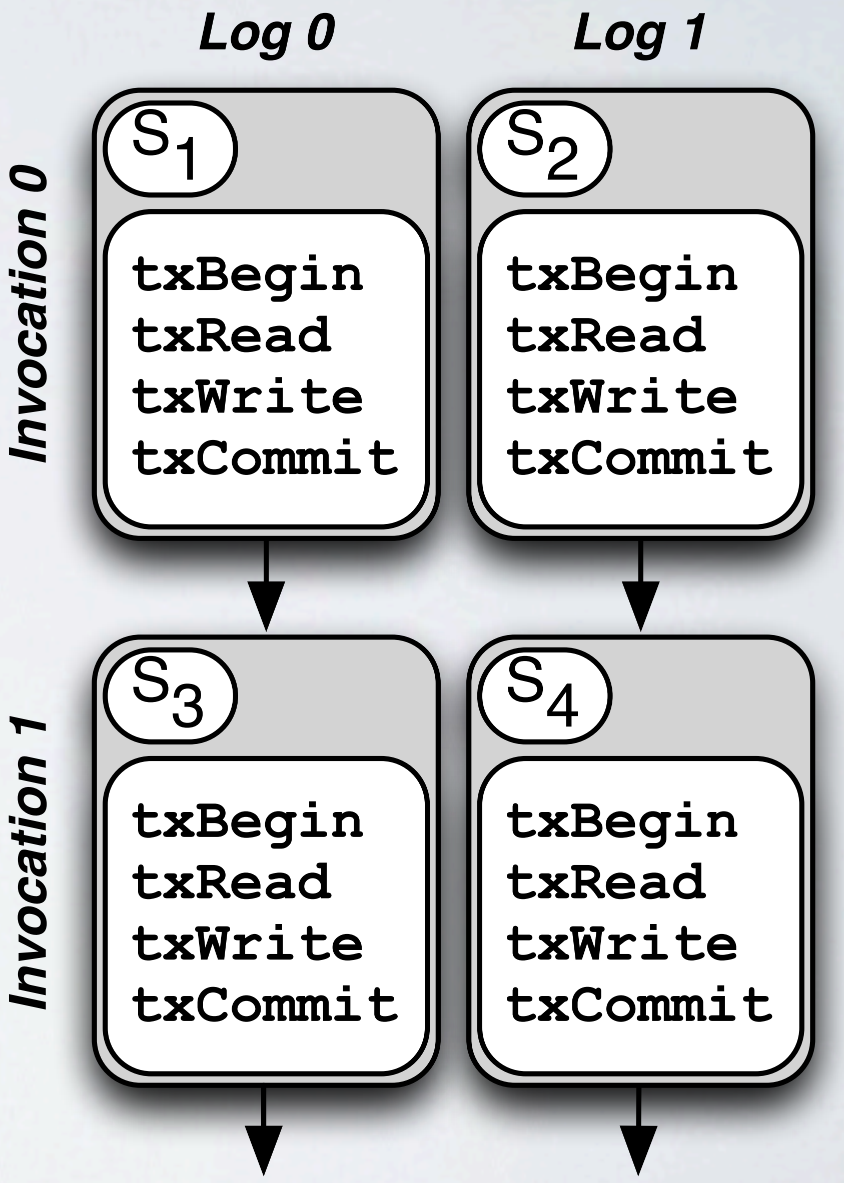
- Log defines *total order* on a sequence of invocations (operations)
  - FIFO sequential execution
- Multiple logs executed in parallel
- Invocations (transactions) applied to *shared memory*
  - *Sequential object* to transform into wait-free *linerizable object*
- Transactions can be *non-terminating*
  - Terminate *correct transactions* within finite number of steps



INVOCATION PROCESSING



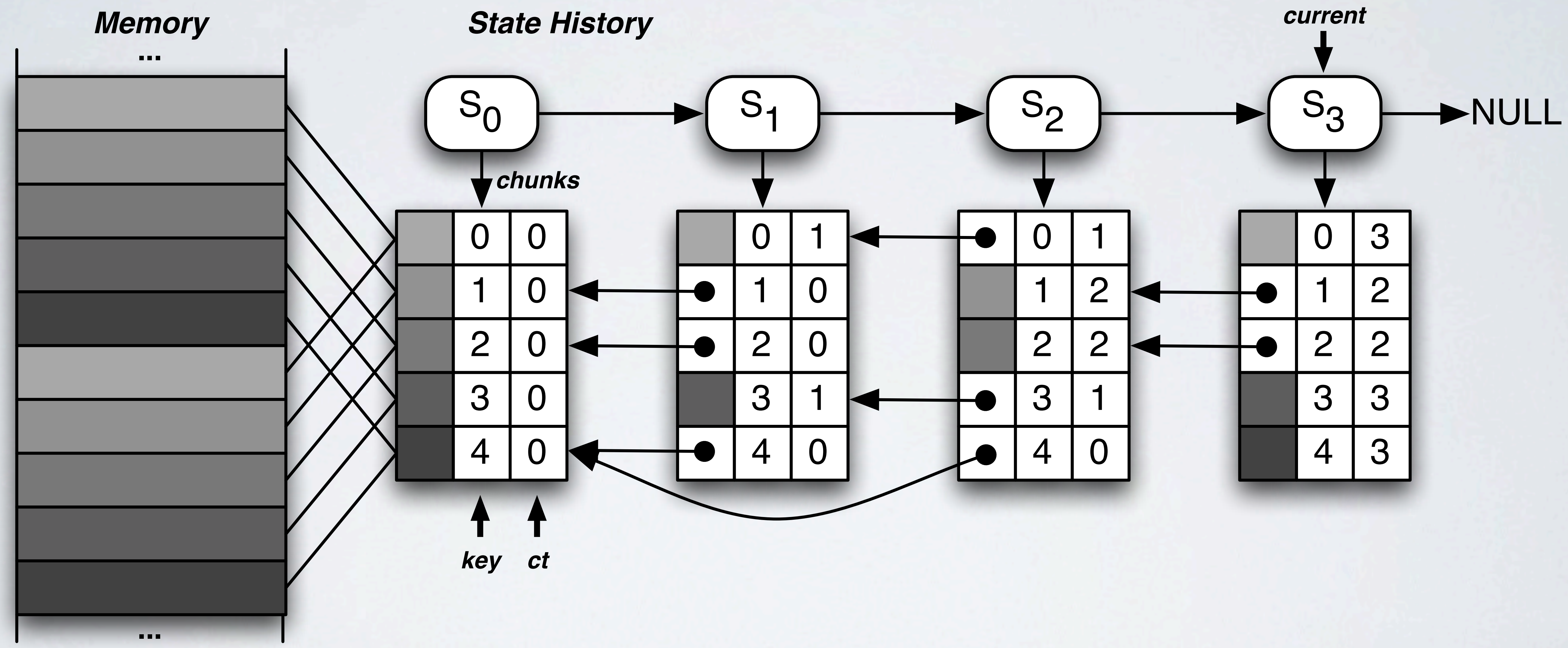
How can one process invocations from *k logs* using *n threads* in a finite number of steps?



UNIVERSAL TRANSACTIONAL MEMORY CONSTRUCTION

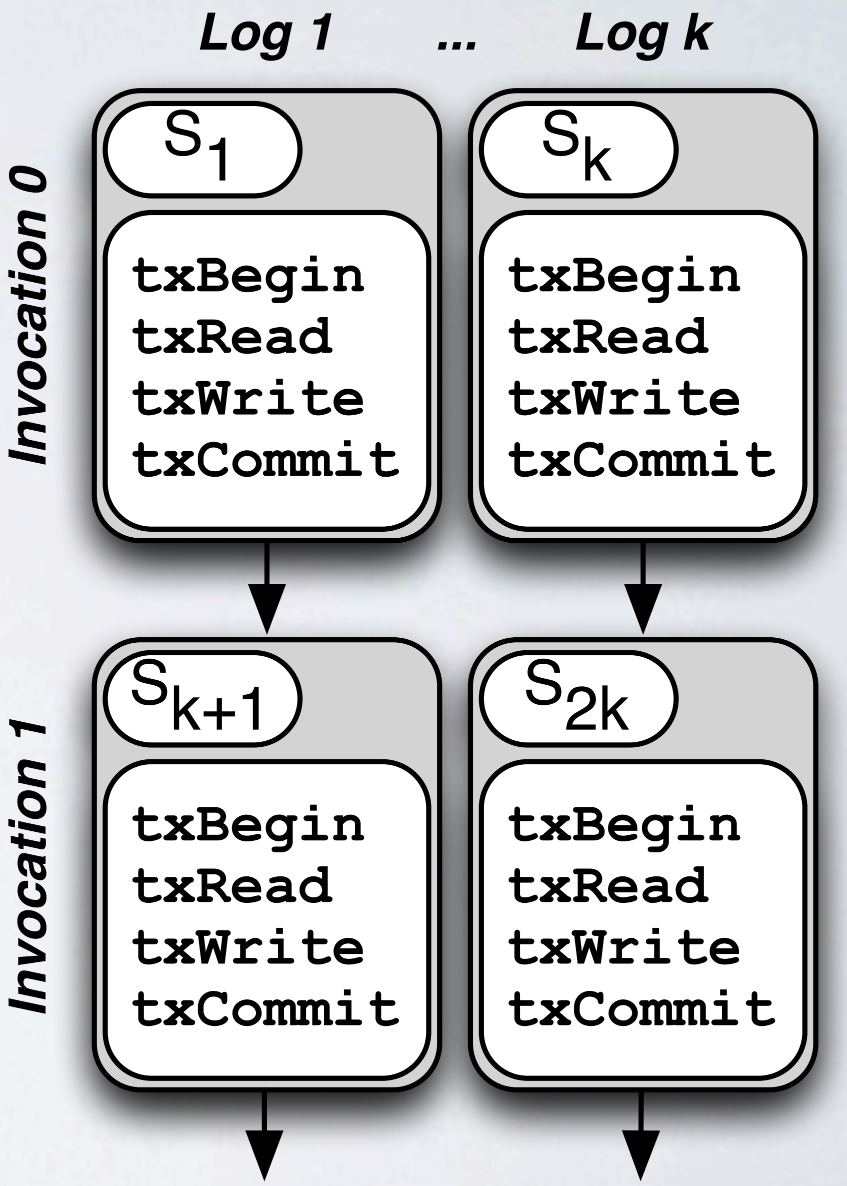
- Universal construction transforms program from one valid state to another valid state
  - Use TM to *isolate modifications* until commit
  - *Schedule* transactions on threads for wait-free progress

STATES


Memory
...
...
State History
current
S_0
S_1
S_2
S_3
NULL
chunks
key
ct
0 0
1 0
2 0
3 0
4 0
0 1
1 0
2 0
3 1
4 0
0 1
1 2
2 2
3 1
4 0
0 3
1 2
2 2
3 3
4 3

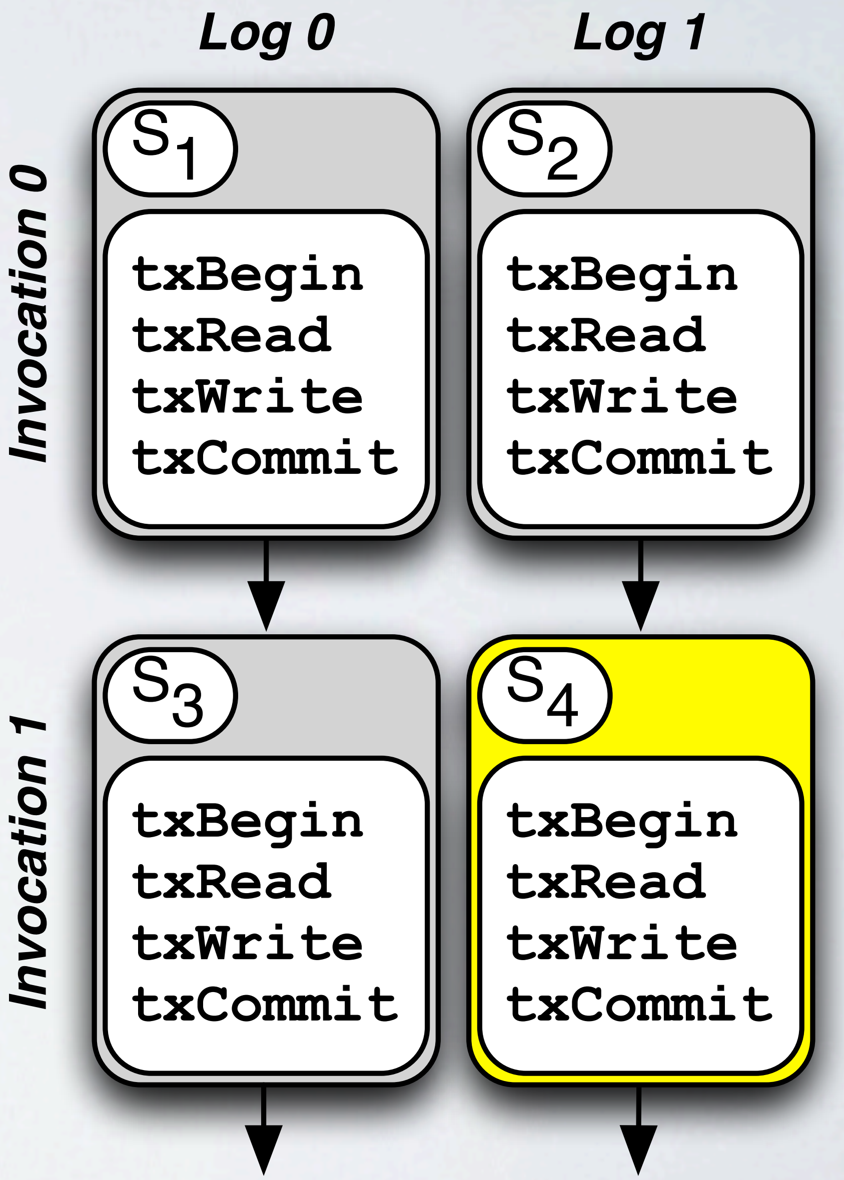
TRANSACTION SCHEDULING ONTO THREADS

- A thread‘s commit time is FAI(*CT*)
- $S_{CT} = \mathtt{f}_{\mathrm{tx}}(S_{CT\text{-}1})$
- $CT\text{-}1 = \mathtt{i} * k + \mathtt{1}$
    - Log $\mathtt{l} = (CT\text{-}1)\ \mathtt{mod}\ k$
    - Invocation $\mathtt{i} = (CT\text{-}1)\ \mathtt{div}\ k$



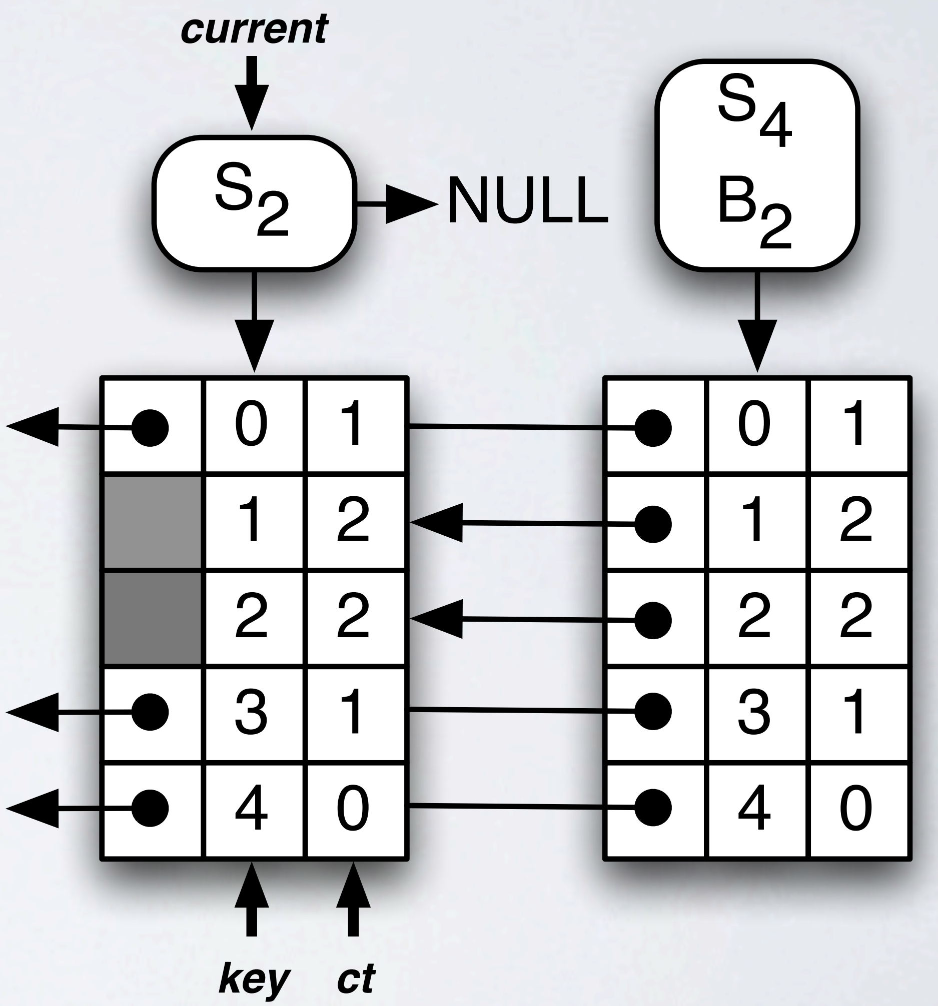
TRANSACTION SCHEDULING ONTO THREADS

- $ct_4 = \mathrm{FAI}(CT_4)$
- $S_4 = \mathtt{f}_{tx}(S_3)$
- $ct_{4-1} = \mathtt{i} * k + \mathtt{1}$
  - Log $\mathtt{l}_1 = 3 \bmod 2$
  - Invocation $\mathtt{i}_1 = 3 \operatorname{\mathtt{div}} 2$



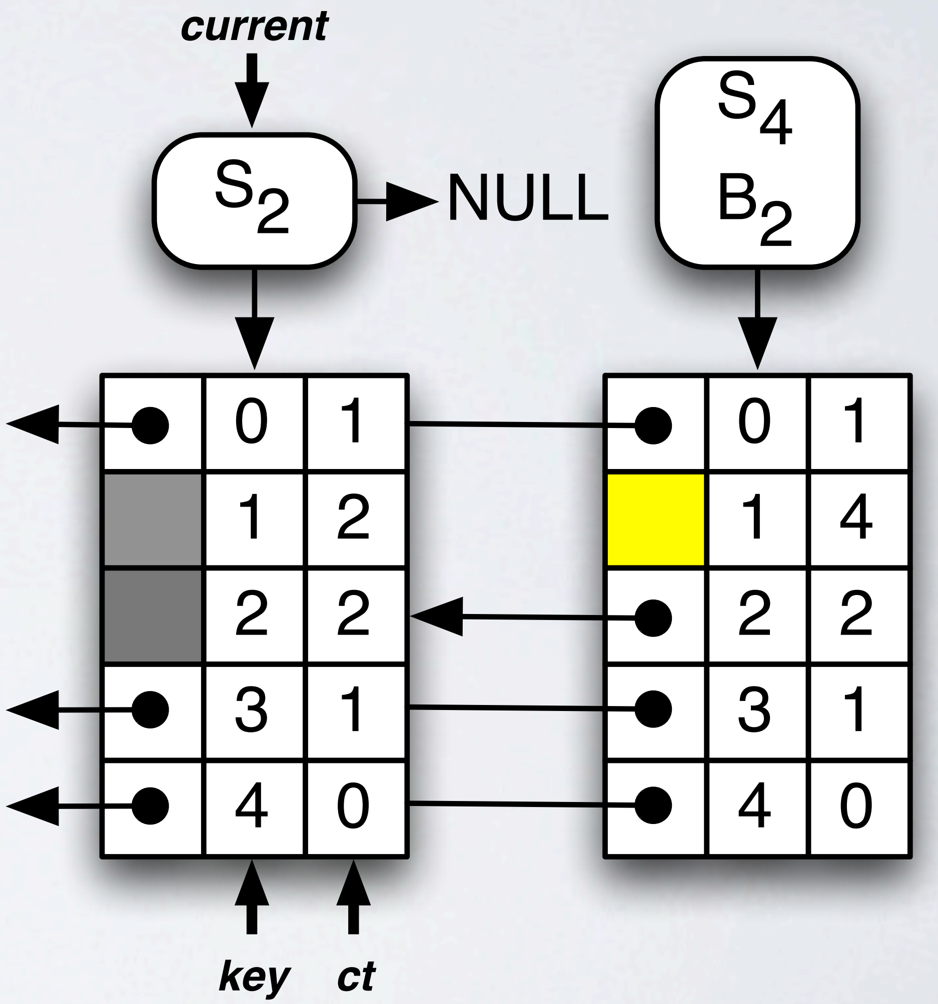
`txBegin(ct)`

- *Clone* current head of state history
  - Chunks are contained only *by reference*
- Set new *commit time*
- Keep *base version*



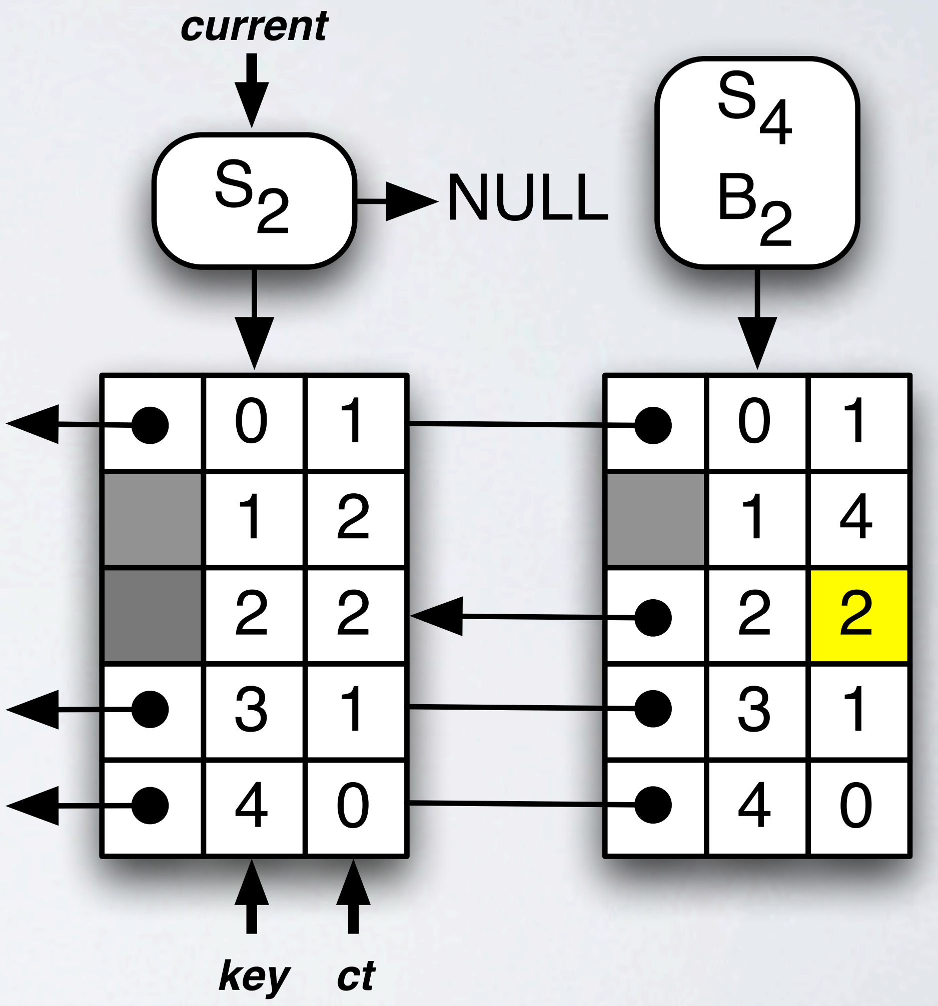
`txWrite(addr, val)`

- *Identify* chunk using hash function
- *Clone* chunk if still reference
- *Update* clone with value



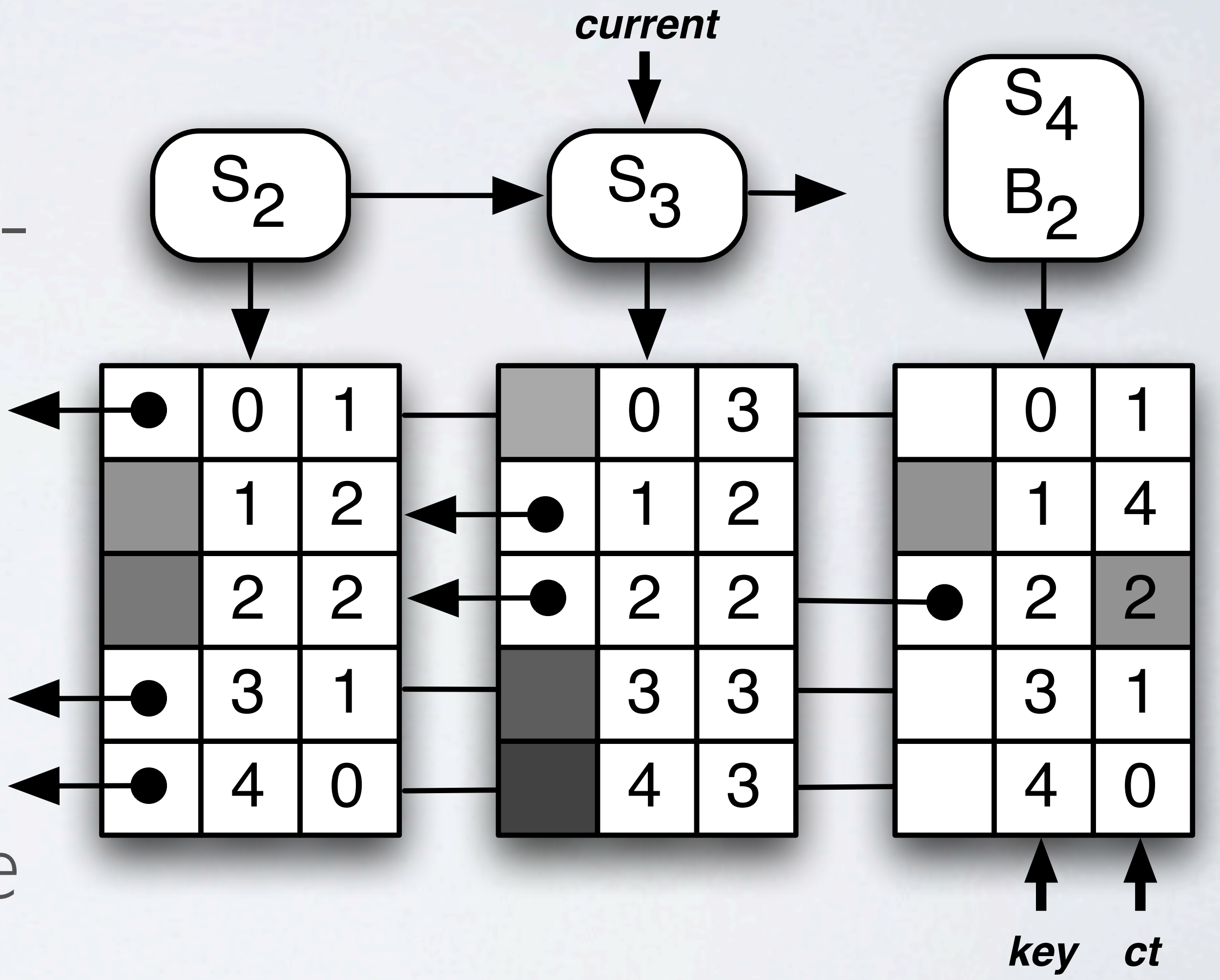
`txRead(addr)`

- *Identify* chunk using hash function
- Keep commit time of chunk in *read-set* if written by predecessor
- Return value of address



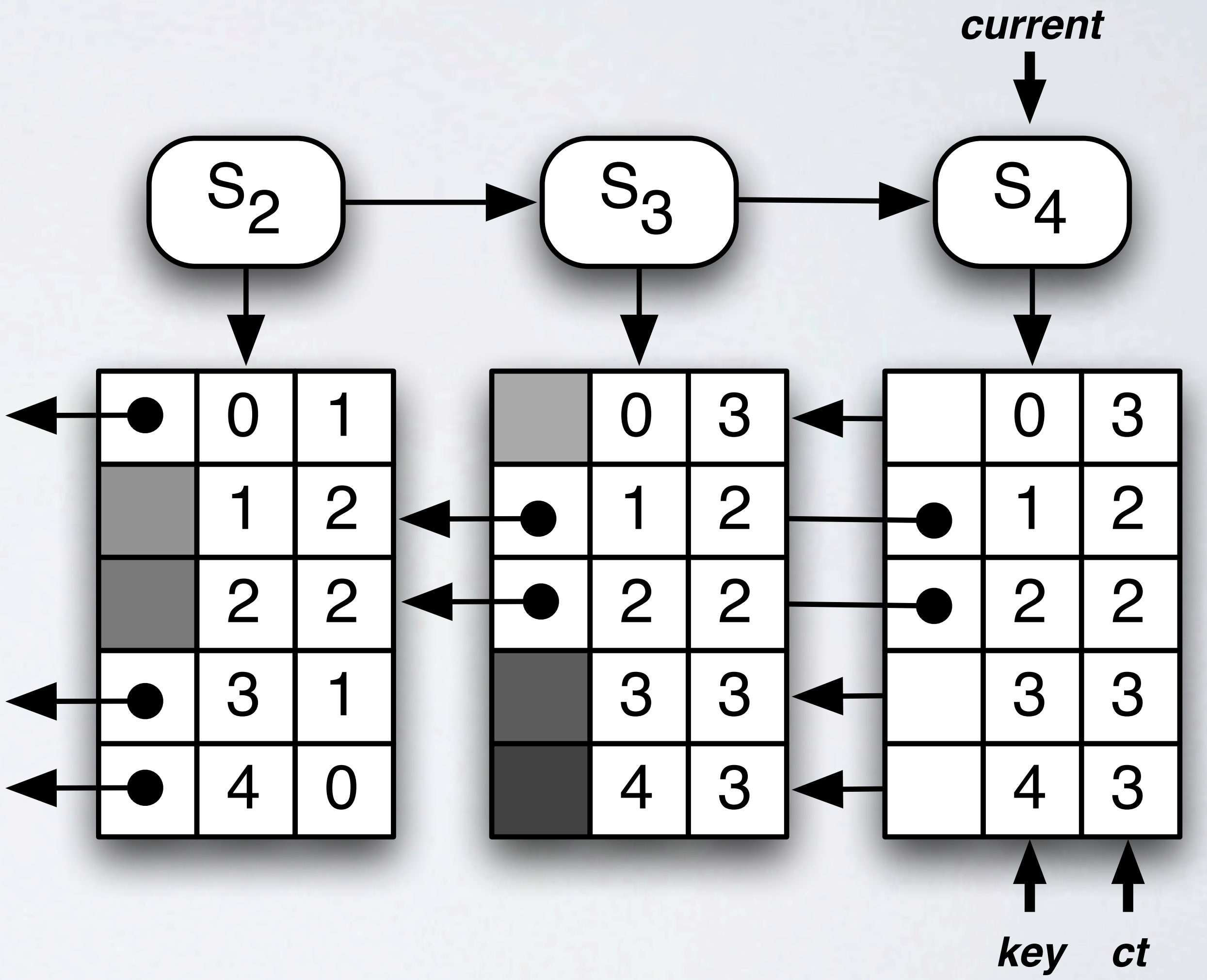
txCommit

- *Wait* until S_{CT-1} is available
- *Validate* against S_{CT-1} that read-set versions unchanged
  - *Failure*: abort and retry
  - *Success*: update chunk references and *append state* to history using CAS

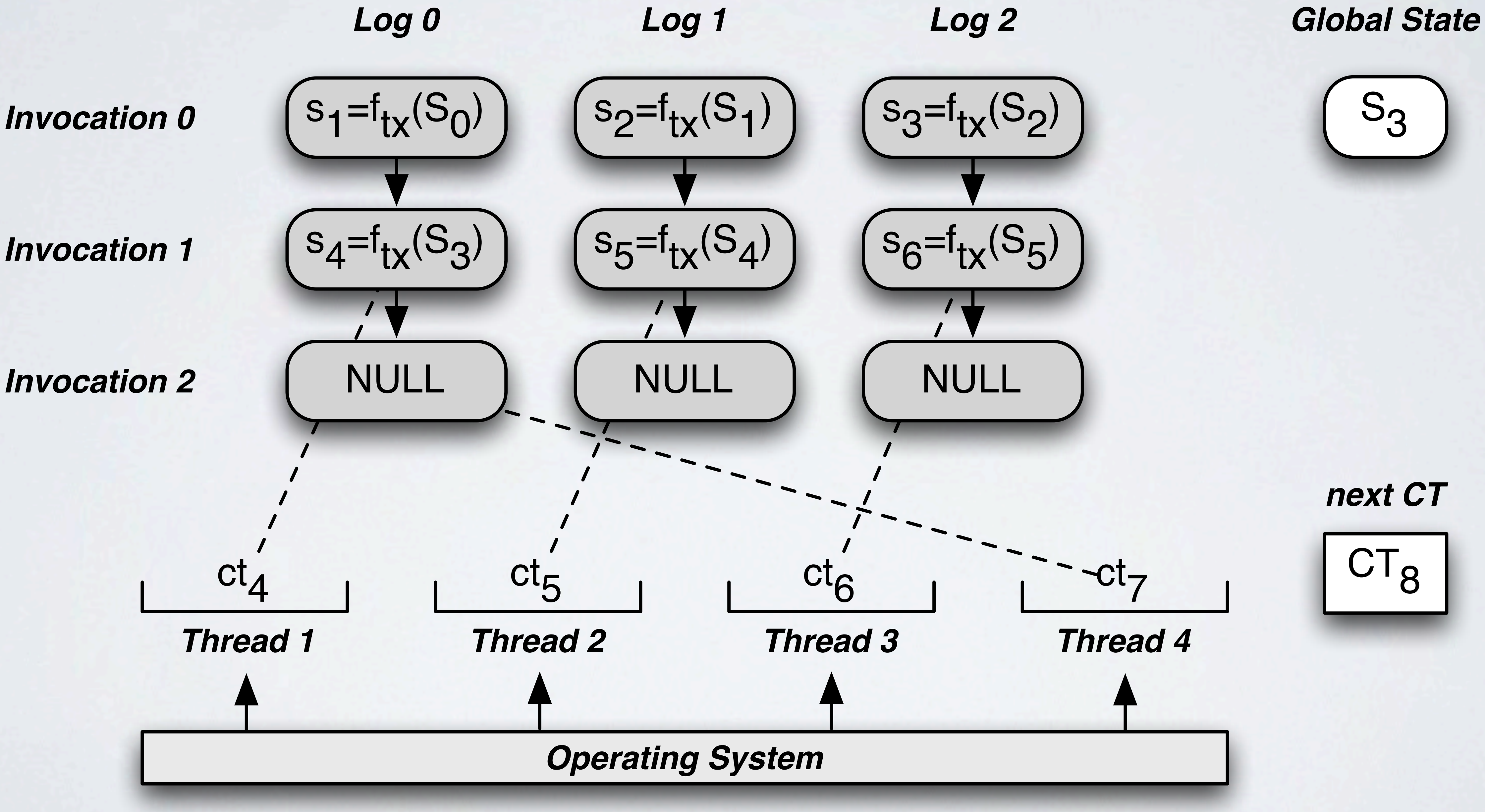


`txCommitProceeding(ct)`

- *Wait* until S_{CT-1} is available
- *Clone* S_{CT-1} and *append* as S_{CT} using CAS

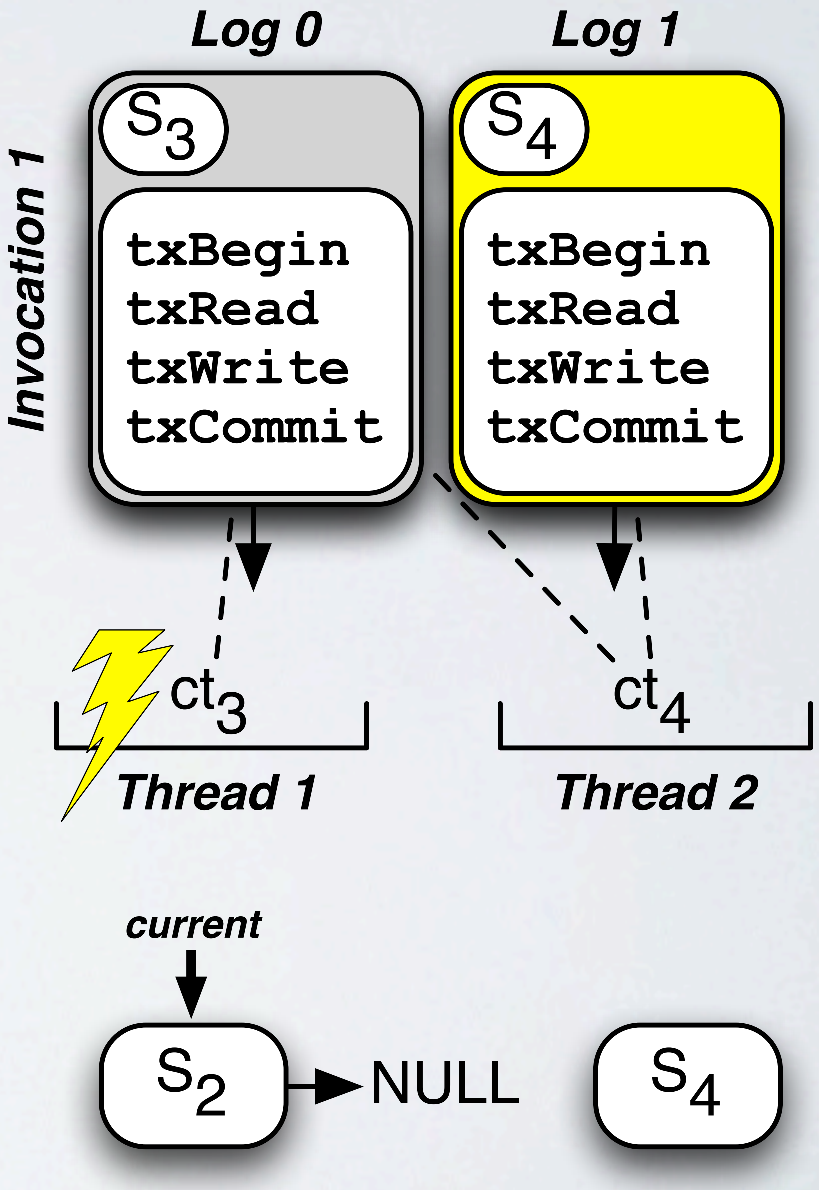


UNIVERSAL CONSTRUCTION FOR THE GOOD CASE



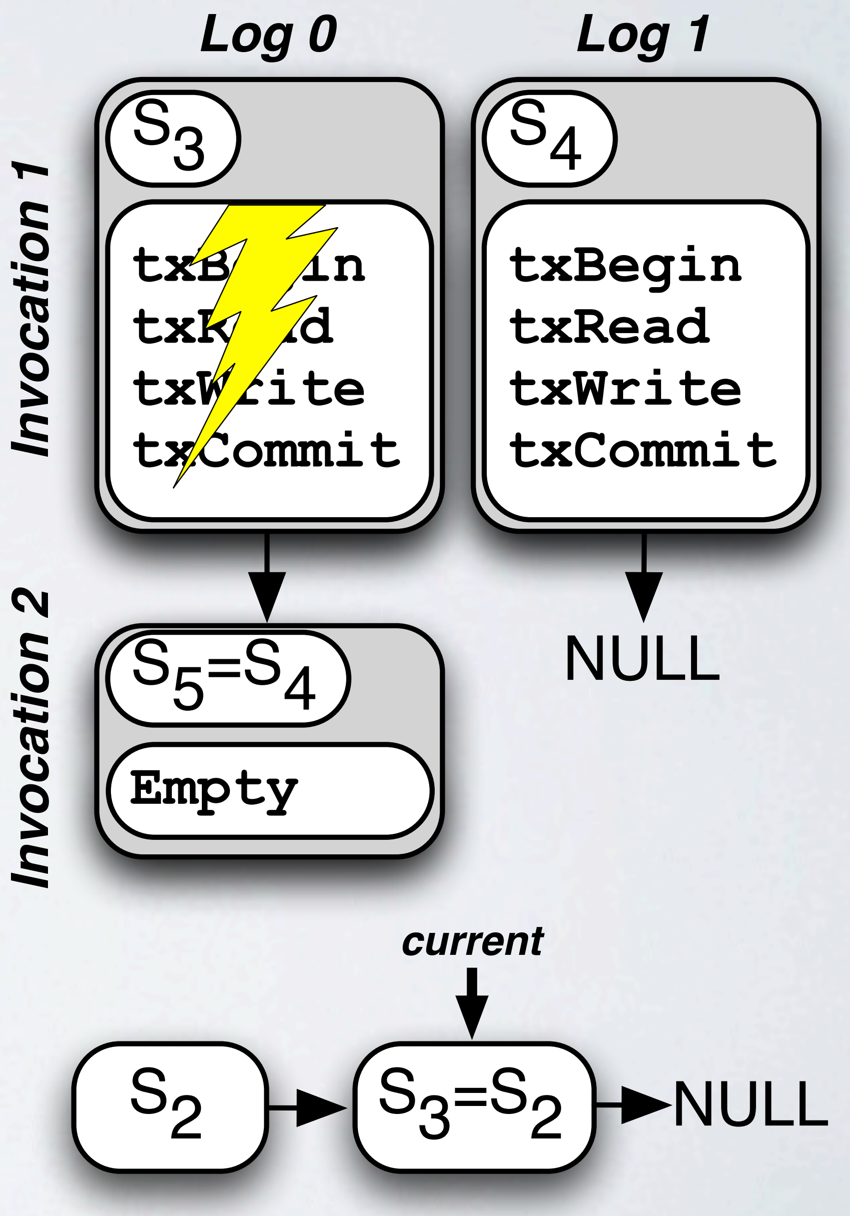
DEALING WITH THREAD CRASHES

- Thread crashes not detectable in AMSM: need *helping*
- Waiting in `txCommit` for S_{CT-1} used for helping
  - Process $S_{CT-1} = f_{tx}(S_{CT-2})$
- Progress as long as one thread survives



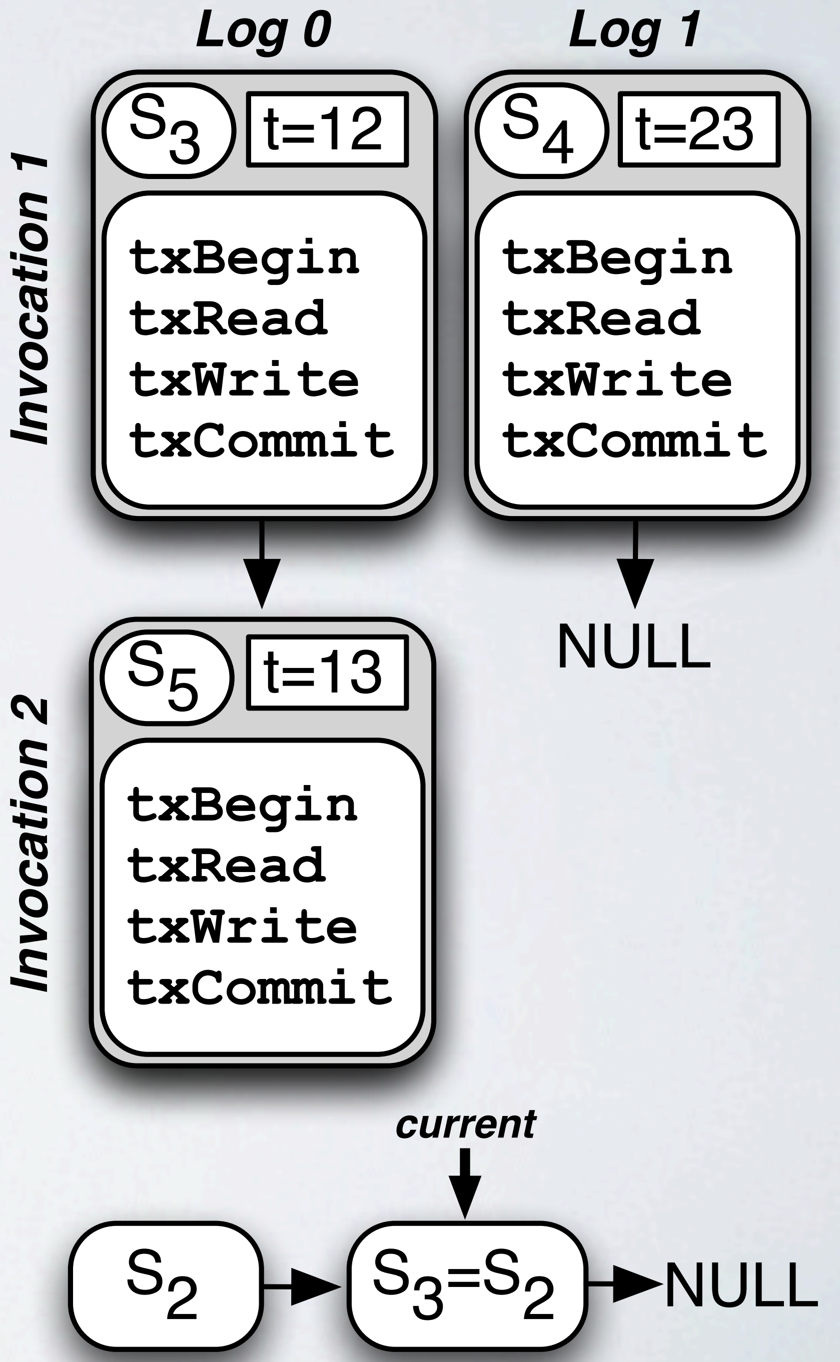
DEALING WITH TRANSACTION CRASHES

- Transaction crashes detectable in AMSM: throw *exception*
  - Considered as *persistent failures*
  - Commit *proceeding* state
  - No further invocations can be added to the log



TOLERATING NON-TERMINATING TRANSACTIONS

- *Non-terminating* transaction: `ftx`(S_{CT-1}) never returns
- Use *performance counter* to assign quota of steps for invocation
  - When exceeded commit proceeding and *retry* with larger quota
- Quota is *unknown* and usually large
  - Bounded number of states (bound memory) means *bound* number of steps for transaction to complete



CONCLUSION

- Wait-free progress under AMSM by:
  - Decoupling of work from threads for helping
  - Isolating changes in transactions to mask failures