ReVivel/O:

Efficient Handling of I/O in Highly-Available Rollback-Recovery Servers

Jun Nakano[✢], Pablo Montesinos, Kourosh Gharachorloo[❄], Josep Torrellas

University of Illinois at Urbana-Champaign
[✢]IBM
[❄]Google

i-acoma group

1867

Building Fault-Tolerant Systems

Building Fault-Tolerant Systems

Redundant
Self-Checking
HW

High Overhead
Small/Null MTTR

HP Nonstop
IBM S/390

Building Fault-Tolerant Systems

Redundant Self-Checking HW

High Overhead
Small/Null MTTR

HP Nonstop
IBM S/390

Plain HW + SW-based Checkpointing

High Overhead
Significant MTTR

KeyKOS
FT Mach

Building Fault-Tolerant Systems

Redundant Self-Checking HW	HW-based High-frequency Checkpointing	Plain HW + SW-based Checkpointing
High Overhead Small/Null MTTR	Low overhead Small MTTR	High Overhead Significant MTTR
HP Nonstop IBM S/390	ReVive, SafetyNet	KeyKOS FT Mach

A glimpse of ReVive (ISCA 2002)

A glimpse of ReVive (ISCA 2002)

- Goal: recover the memory state of a shared-memory machine in < 1s

A glimpse of ReVive (ISCA 2002)

- Goal: recover the memory state of a shared-memory machine in < 1s
- HW-assisted very frequent checkpoint (20ms < *T* < 100ms)
  - During checkpoints:
    - Write-back dirty data from caches
    - Main memory is the checkpoint state
  - Between checkpoints:
    - HW logs overwritten data in memory when is modified for the first time

A glimpse of ReVive (ISCA 2002)

- Goal: recover the memory state of a shared-memory machine in < 1s
- HW-assisted very frequent checkpoint (20ms < *T* < 100ms)
  - During checkpoints:
    - Write-back dirty data from caches
    - Main memory is the checkpoint state
  - Between checkpoints:
    - HW logs overwritten data in memory when is modified for the first time
- Entire main memory protected by distributed parity
  - Like RAID-5, but in memory
  - Can tolerate the loss of a node

Building Fault-Tolerant Systems

Redundant Self-Checking HW	HW-based High-frequency Checkpointing	Plain HW + SW-based Checkpointing
High Overhead Small/Null MTTR	Low overhead Small MTTR	High Overhead Significant MTTR
HP Nonstop IBM S/390	ReVive, SafetyNet	KeyKOS FT Mach

Building Fault-Tolerant Systems

Redundant Self-Checking HW	HW-based High-frequency Checkpointing	Plain HW + SW-based Checkpointing
Expensive	Low overhead Small MTTR	High Overhead Significant MTTR
	ReVive, SafetyNet	KeyKOS FT Mach

Building Fault-Tolerant Systems

Redundant Self-Checking HW	HW-based High-frequency Checkpointing	Plain HW + SW-based Checkpointing
Expensive	Low overhead Small MTTR ReVive, SafetyNet	Significant MTTR

Building Fault-Tolerant Systems

Redundant Self-Checking HW	HW-based High-frequency Checkpointing	Plain HW + SW-based Checkpointing
Expensive	Modest HW No I/O support	Significant MTTR

Building Fault-Tolerant Systems

Redundant Self-Checking HW	HW-based High-frequency Checkpointing	Plain HW + SW-based Checkpointing
Expensive	Modest HW No I/O support (until today)	Significant MTTR

Why is I/O undo/redo hard?

Why is I/O undo/redo hard?

- Output Commit Problem:

Why is I/O undo/redo hard?

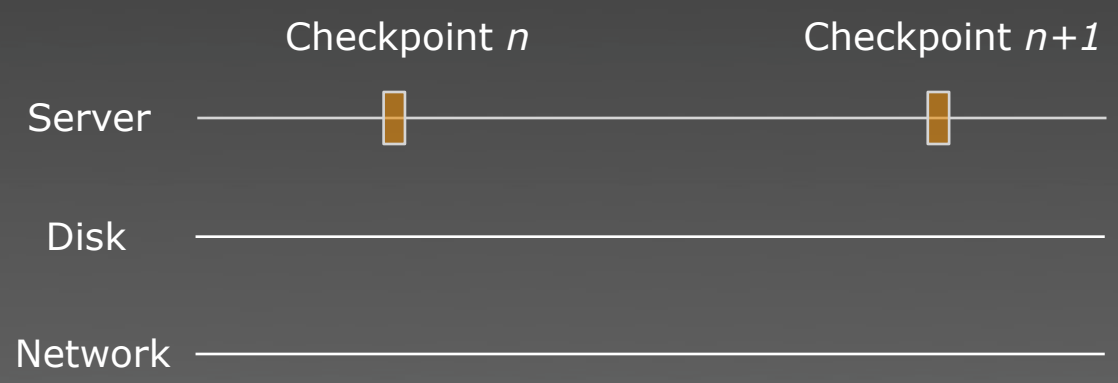
- Output Commit Problem:
  - An output to the external world cannot be rolled back

Why is I/O undo/redo hard?

- Output Commit Problem:
  - An output to the external world cannot be rolled back

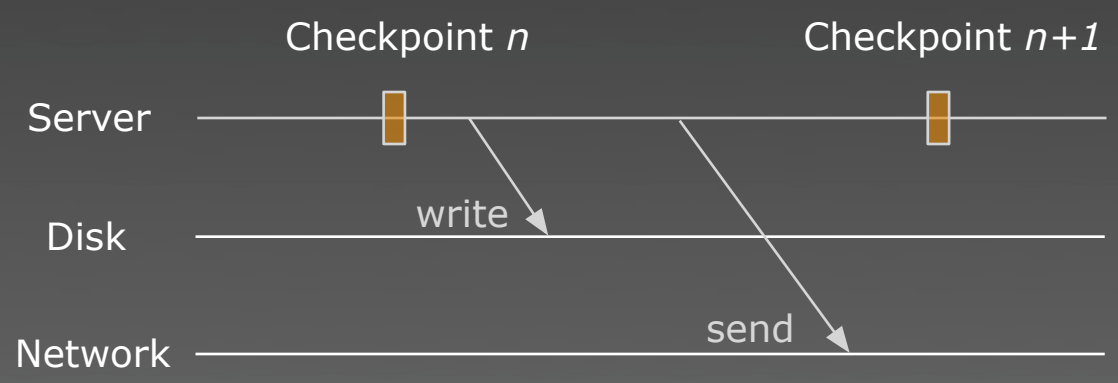
Why is I/O undo/redo hard?

- Output Commit Problem:
  - An output to the external world cannot be rolled back



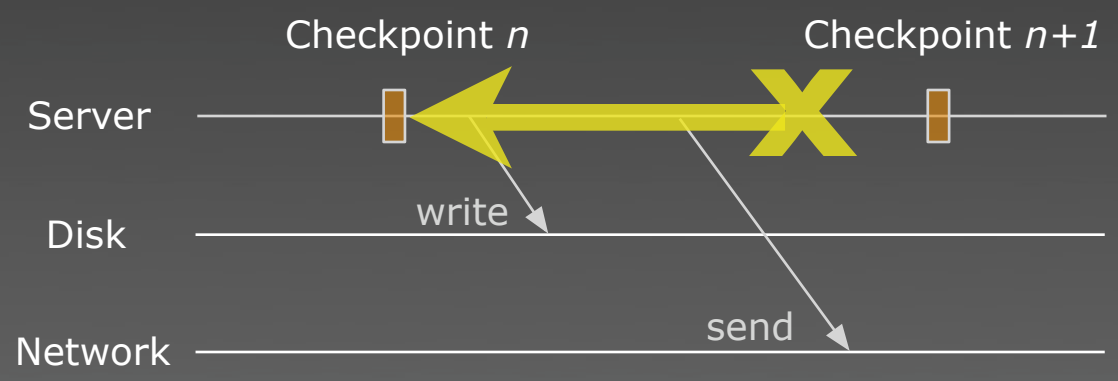
Why is I/O undo/redo hard?

- Output Commit Problem:
  - An output to the external world cannot be rolled back



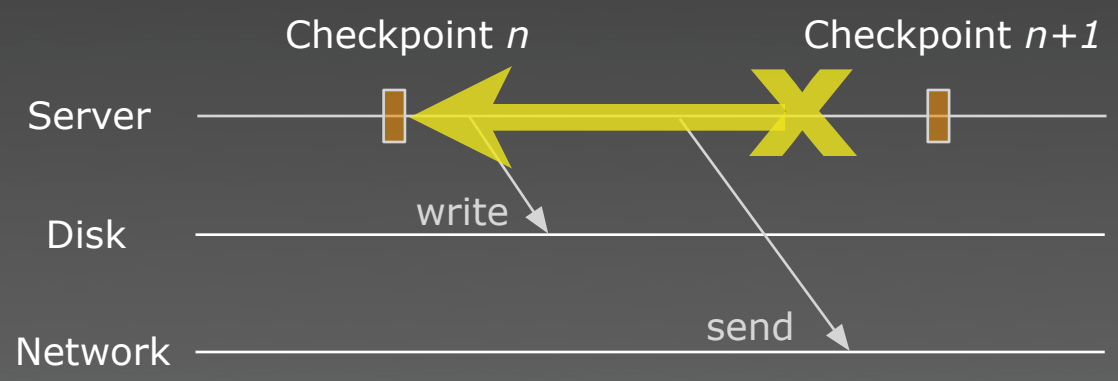
Why is I/O undo/redo hard?

- Output Commit Problem:
  - An output to the external world cannot be rolled back



Why is I/O undo/redo hard?

- Output Commit Problem:
  - An output to the external world cannot be rolled back:



  - How do you undo the I/O?

Contribution: ReViveI/O

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
- Low overhead during fault-free operation (<1% over ReVive)

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
- Low overhead during fault-free operation (<1% over ReVive)
- Fast recovery (< 1s even if processor lost)

Contribution: ReViveI/O

- Support I/O undo/redo in rollback-recovery SMP servers
- Targeted to throughput-oriented workloads
  - Disk and network only
- Transparent to OS and applications
- Low overhead during fault-free operation (<1% over ReVive)
- Fast recovery (< 1s even if processor lost)
- Small impact on hardware cost

Approach: Pseudo-Device Driver

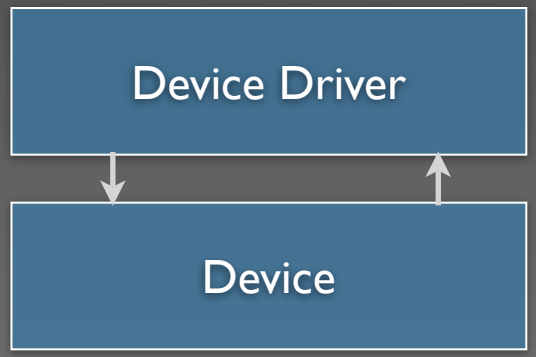
Masubuchi et al, FTCS-97

Approach: Pseudo-Device Driver

Device

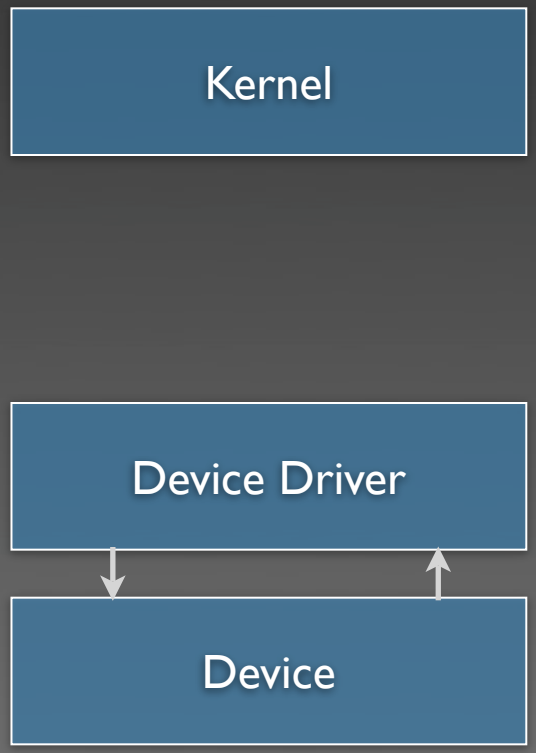
Masubuchi et al, FTCS-97

Approach: Pseudo-Device Driver



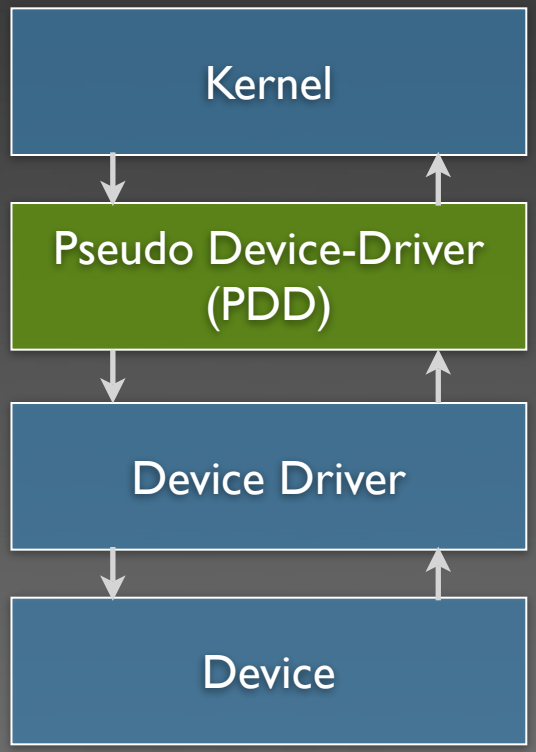
Masubuchi et al, FTCS-97

Approach: Pseudo-Device Driver



Masubuchi et al, FTCS-97

Approach: Pseudo-Device Driver

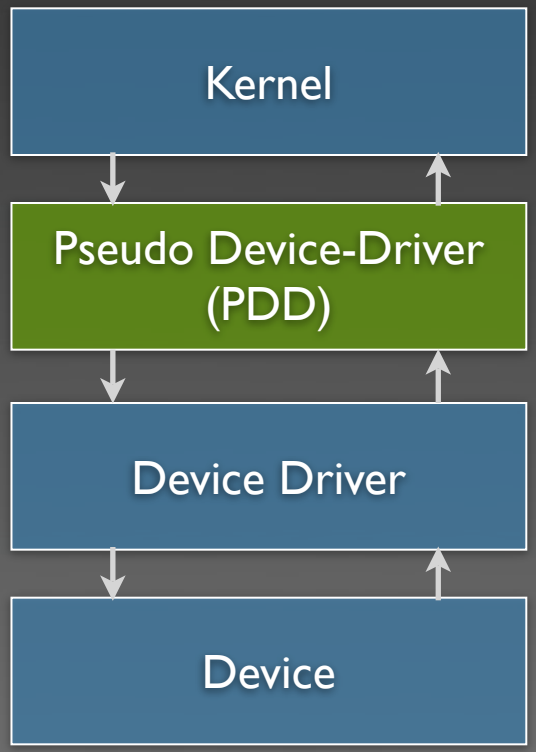


Masubuchi et al, FTCS-97

Approach: Pseudo-Device Driver

- Delays outputs until next checkpoint
- Commits outputs after next checkpoint

- No need to modify OS nor applications
- Network and Disk PDDs



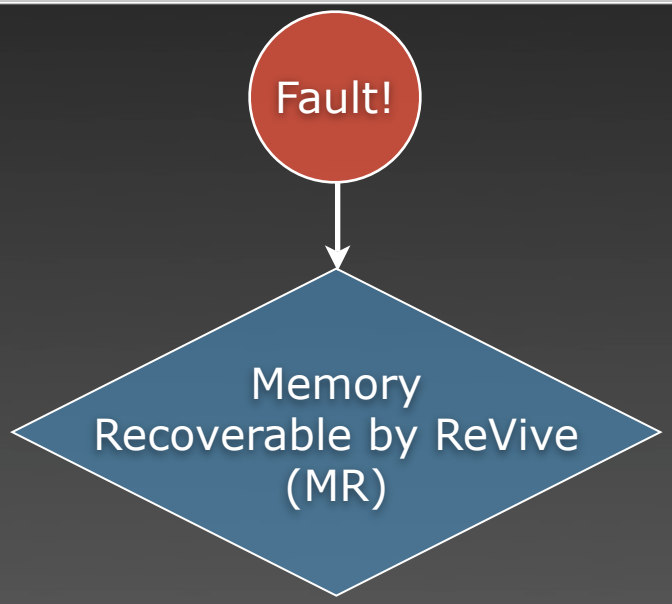
Masubuchi et al, FTCS-97

Fault Model

Fault Model

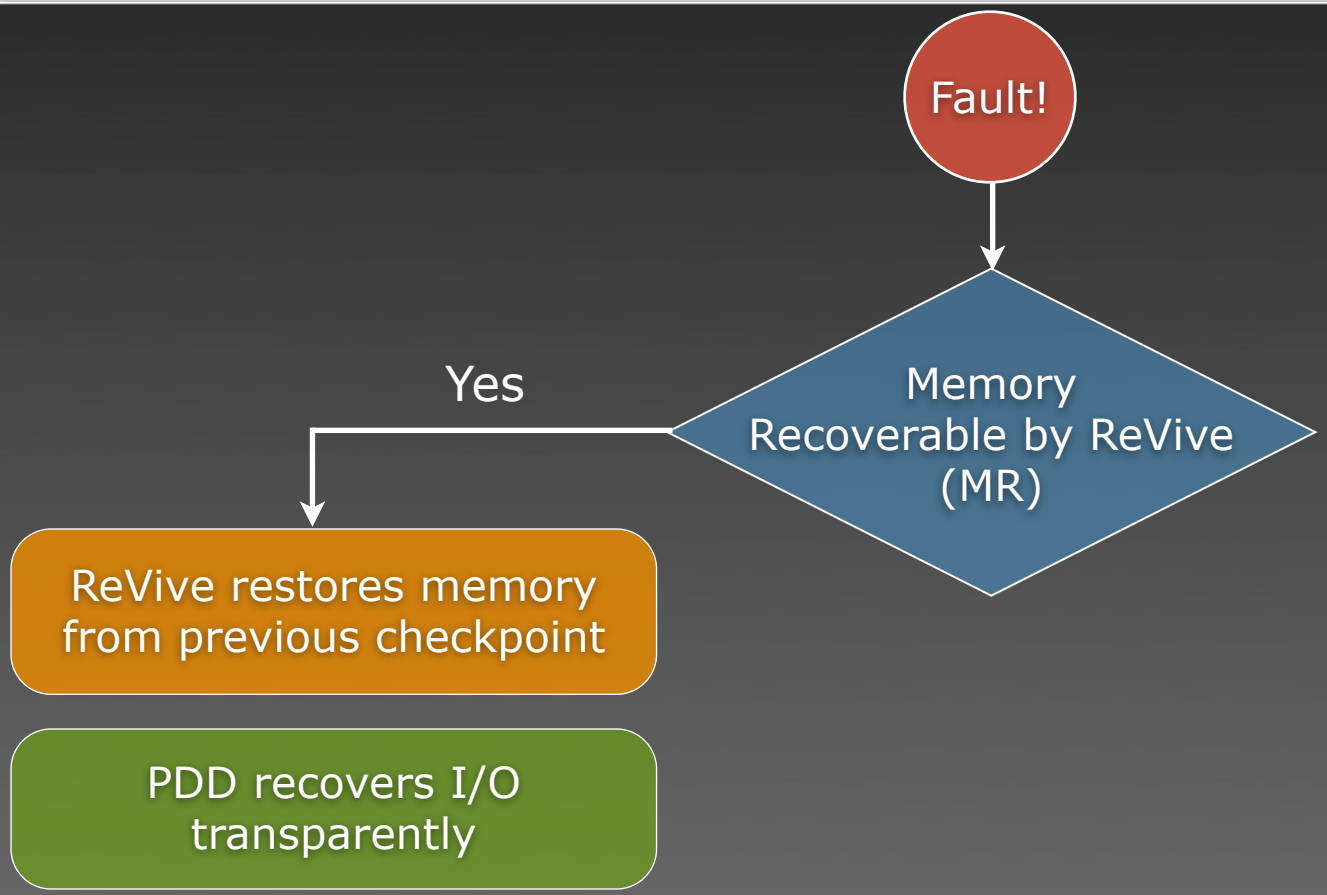
Fault!

Fault Model



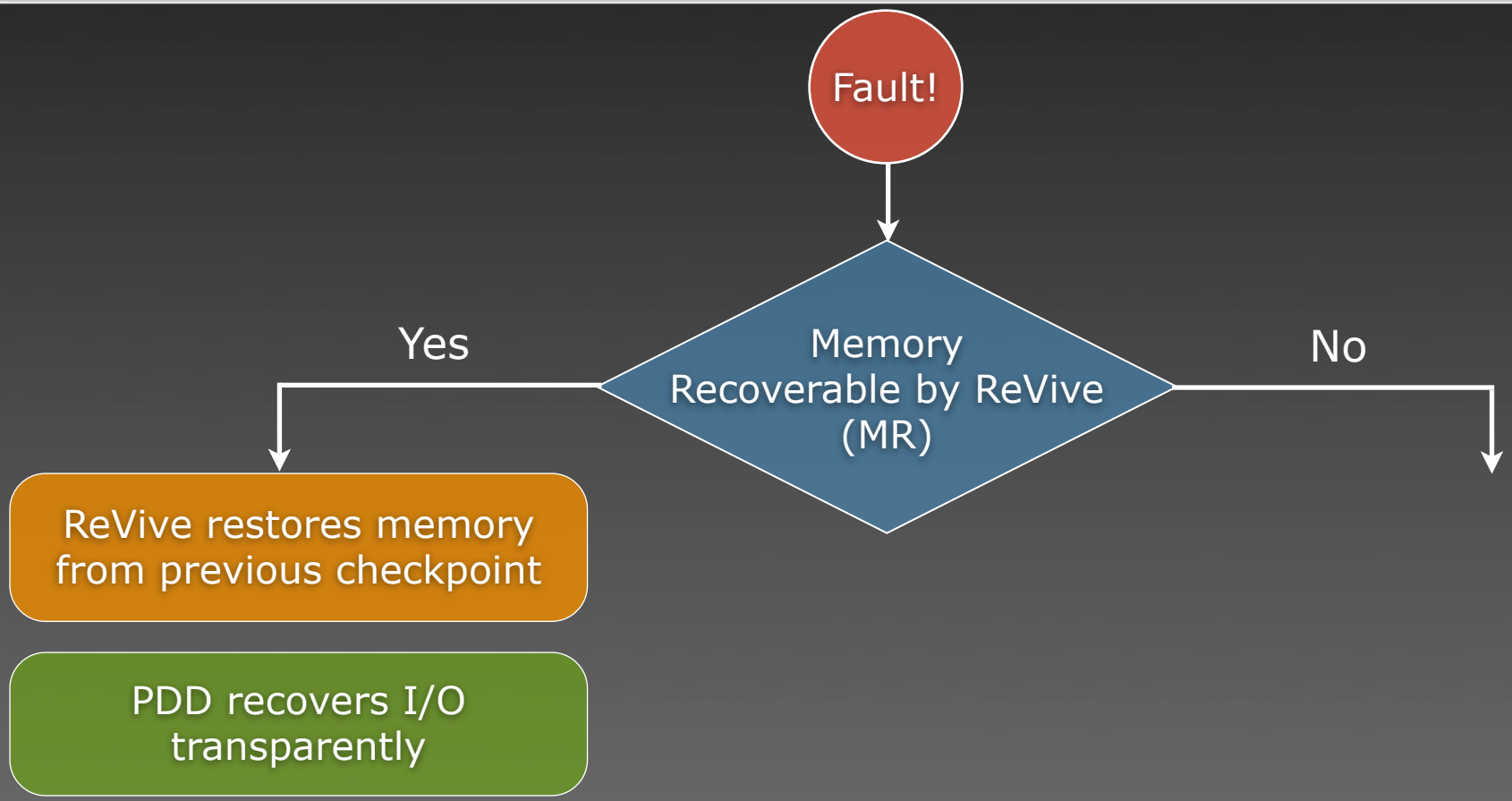
MR=Transients or 1 permanent with at most 1 node loss

Fault Model



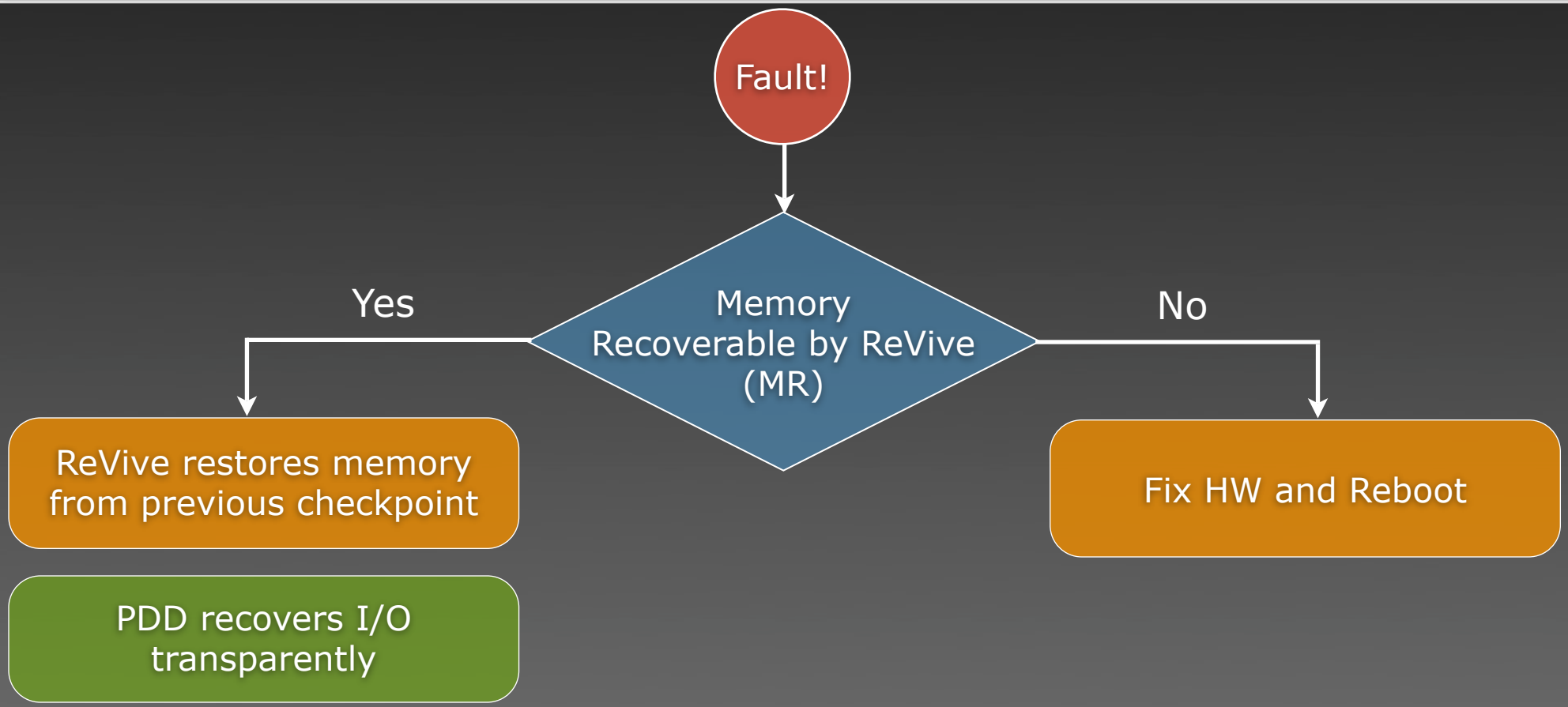
MR=Transients or 1 permanent with at most 1 node loss

Fault Model



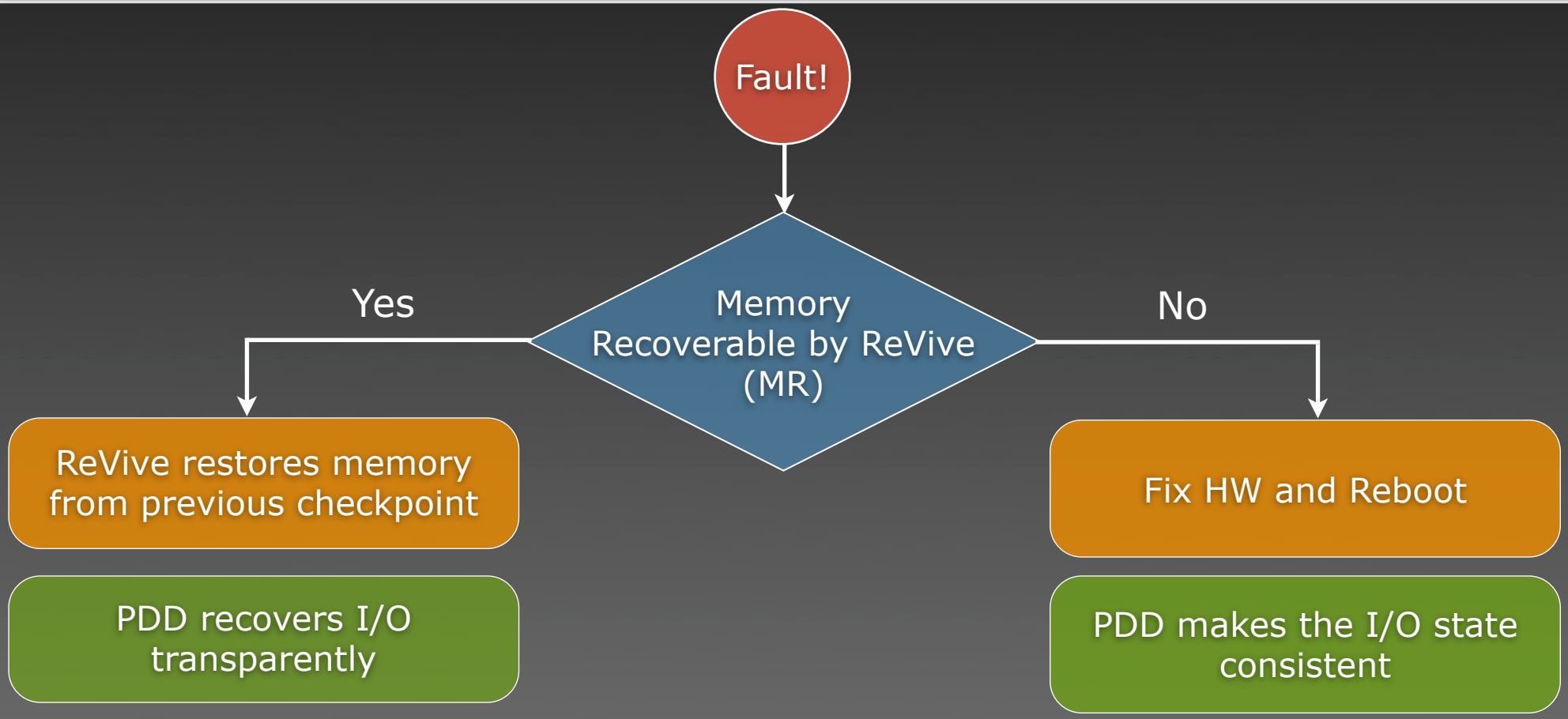
MR=Transients or 1 permanent with at most 1 node loss

Fault Model



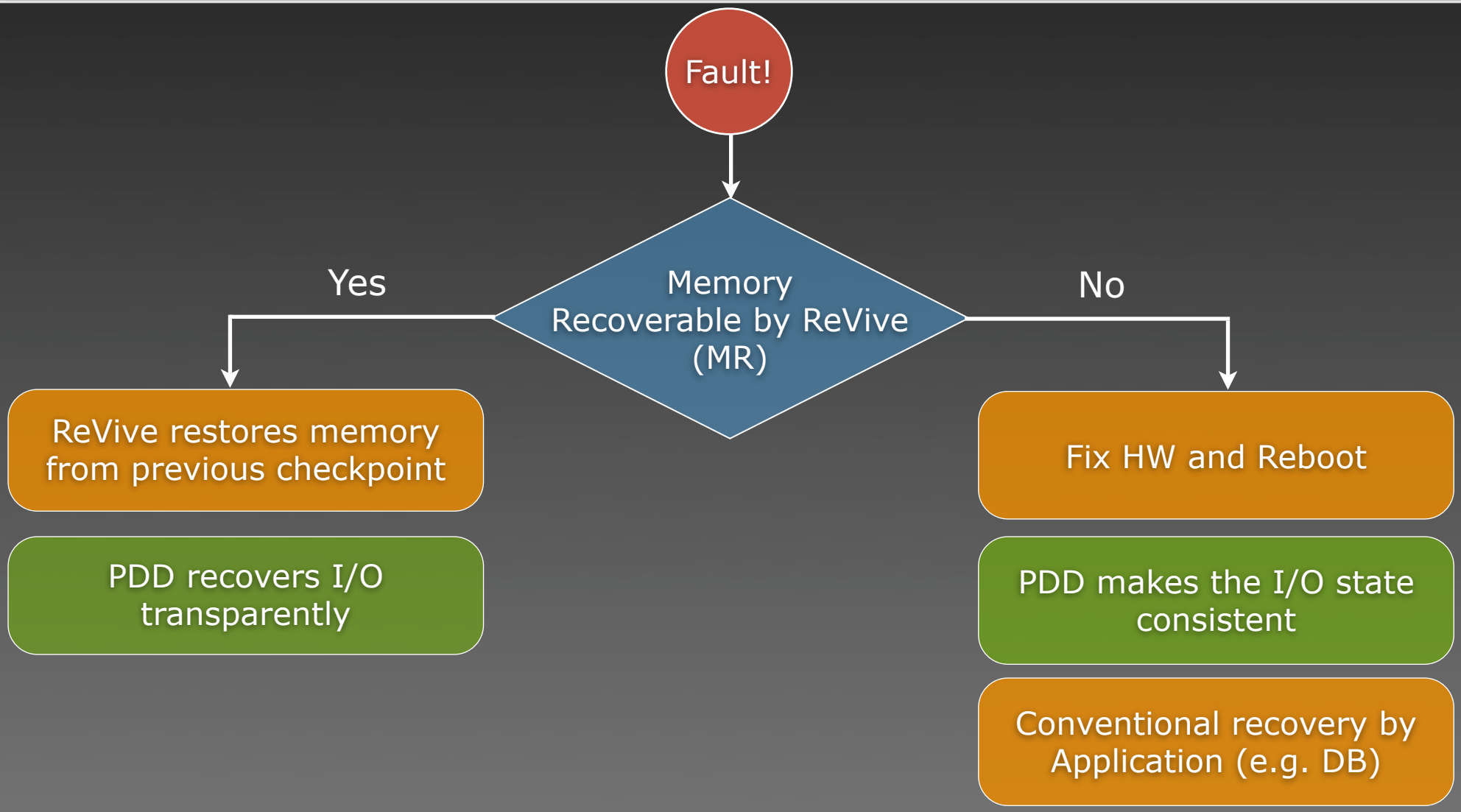
MR=Transients or 1 permanent with at most 1 node loss

Fault Model



MR=Transients or 1 permanent with at most 1 node loss

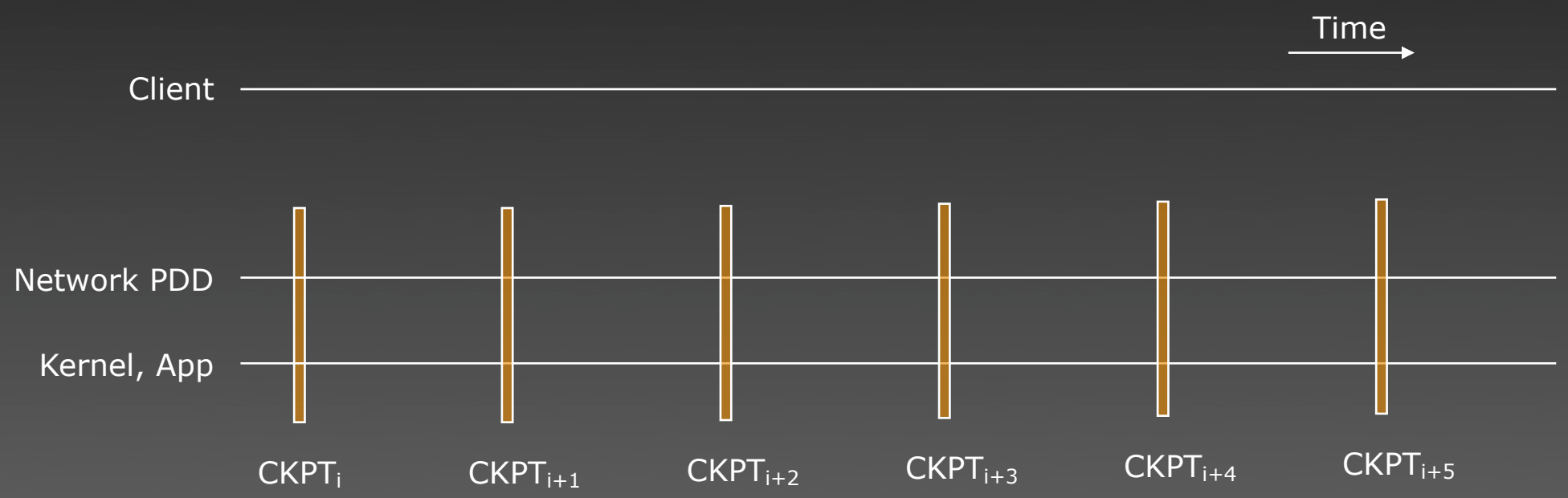
Fault Model



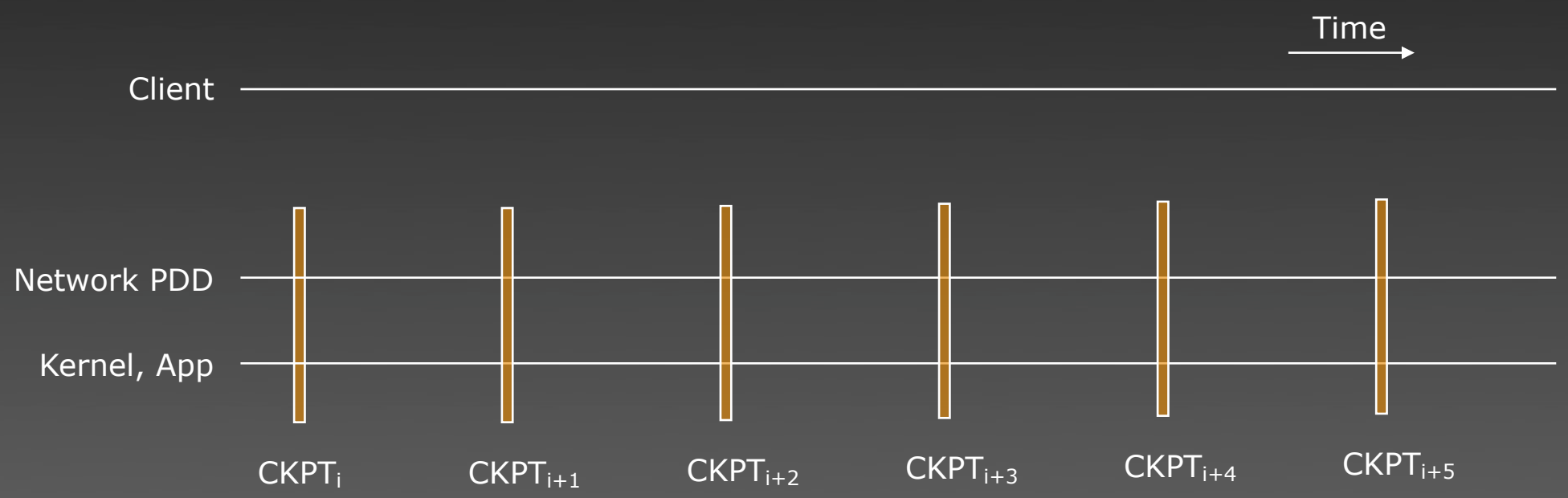
MR=Transients or 1 permanent with at most 1 node loss

Network PDD and TCP

Network PDD and TCP

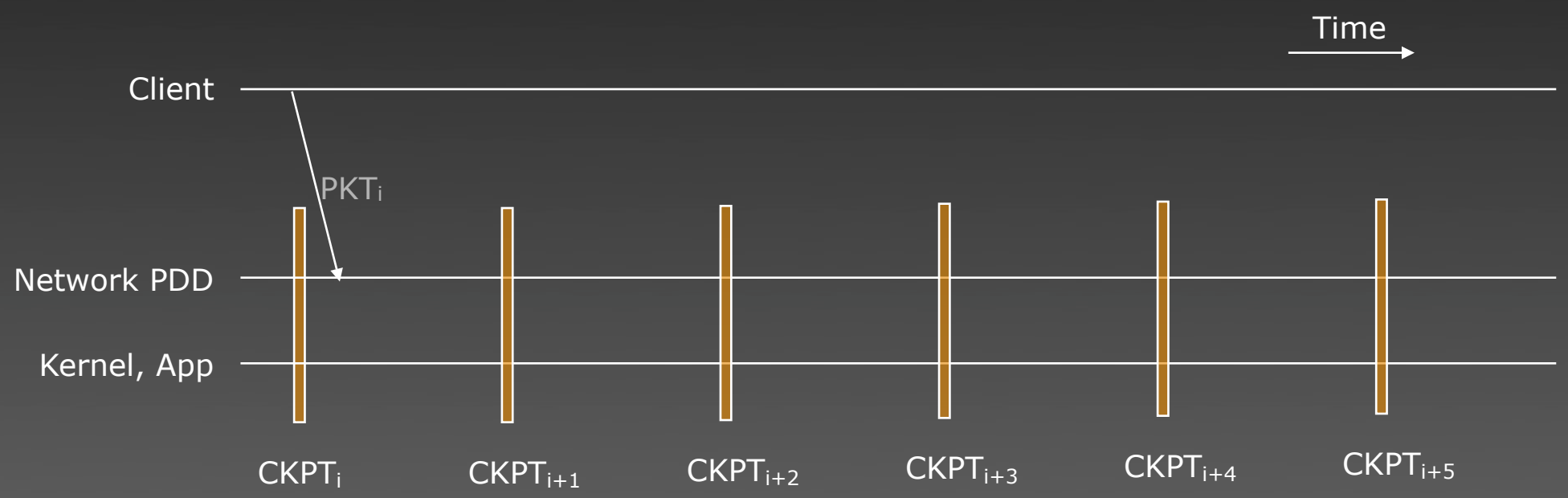


Network PDD and TCP



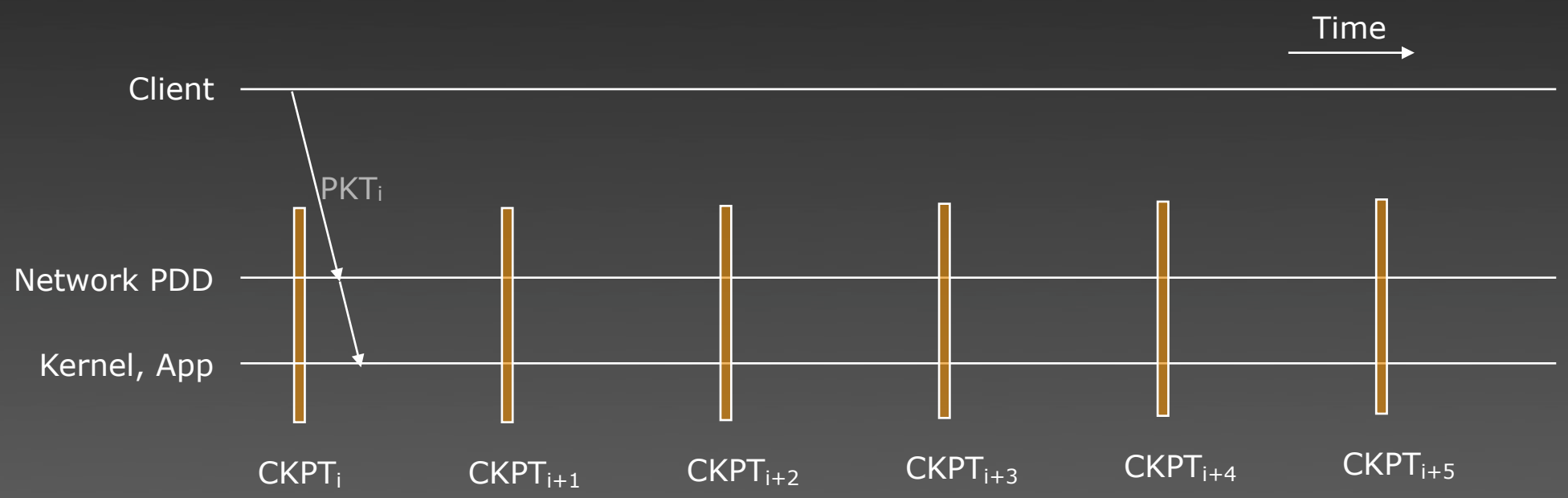
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



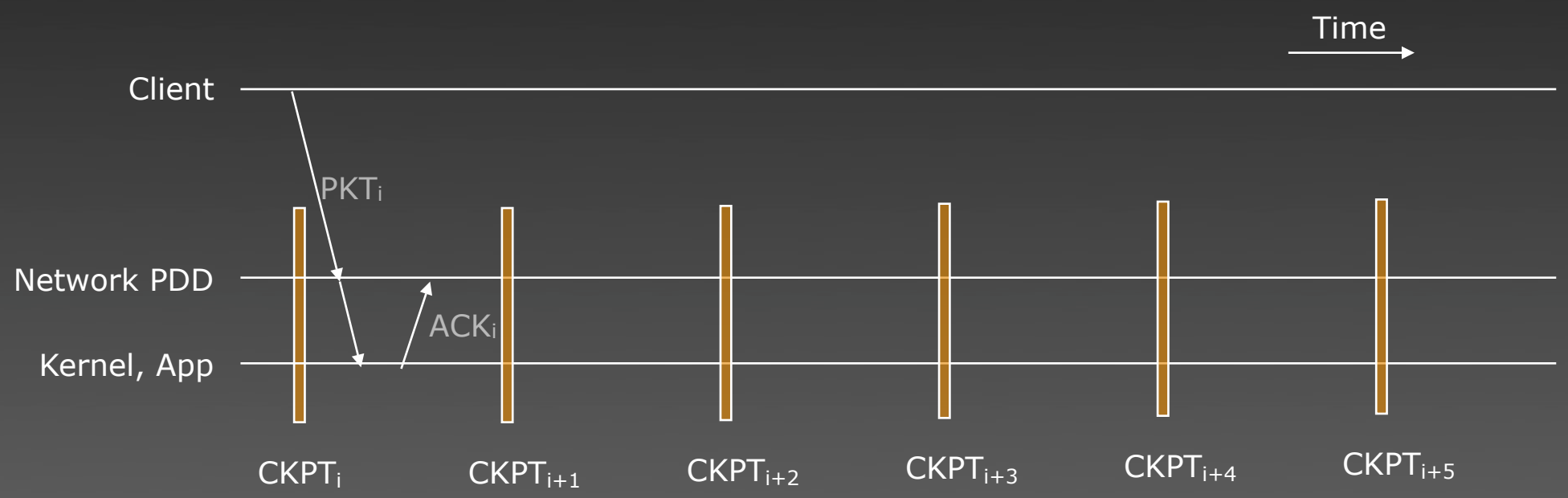
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



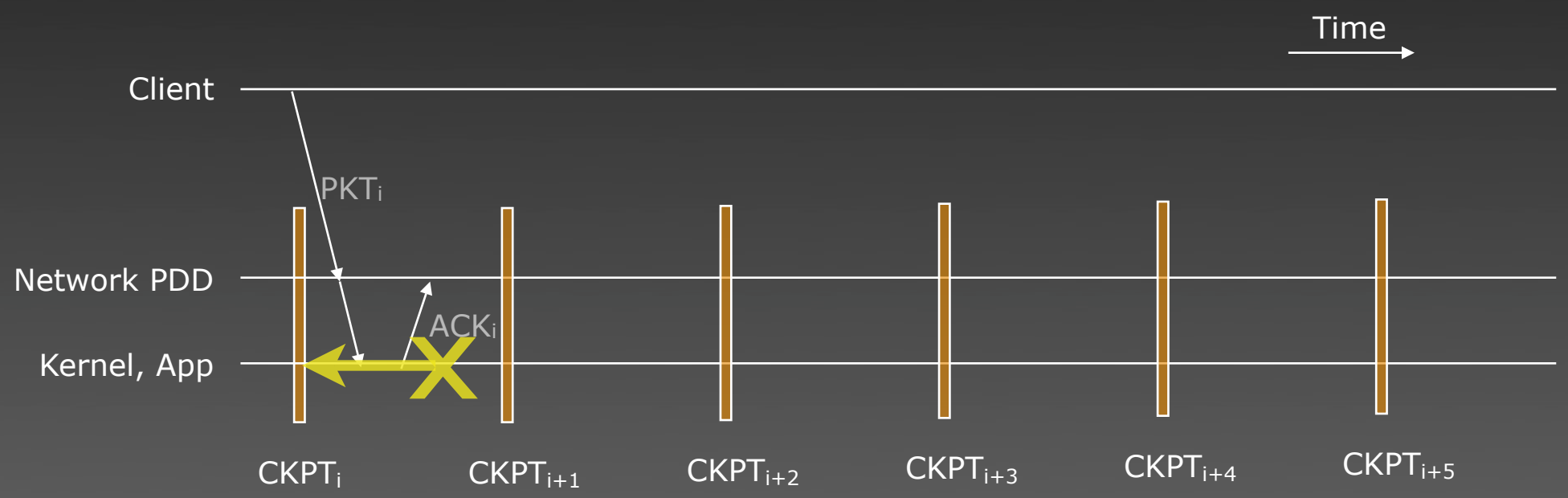
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



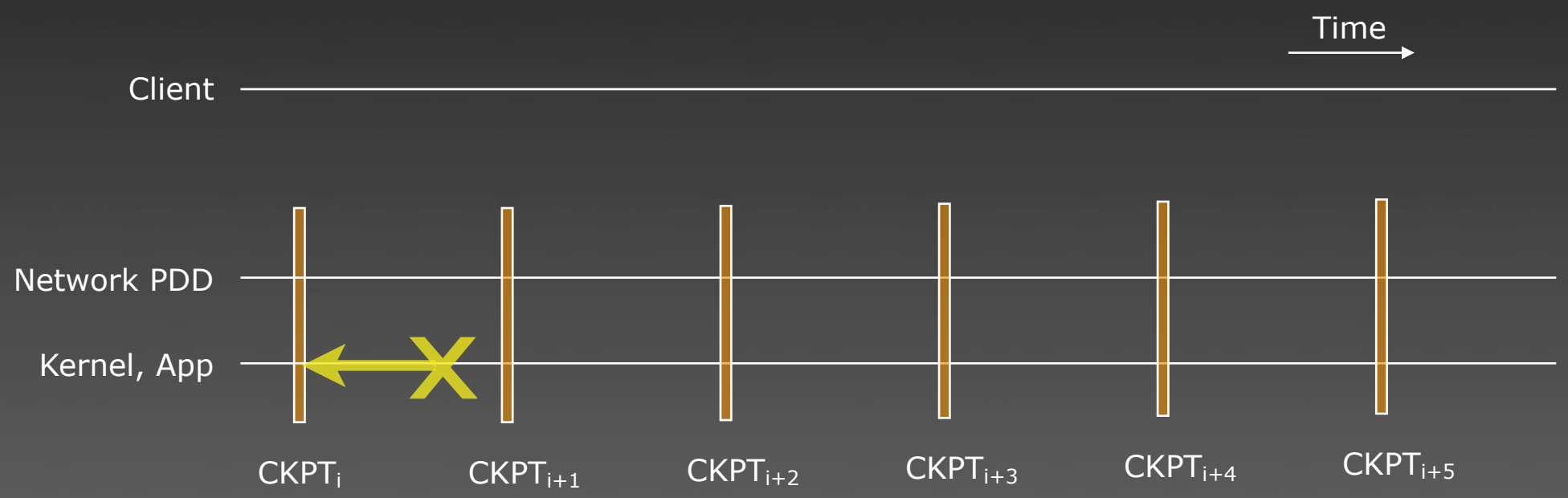
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



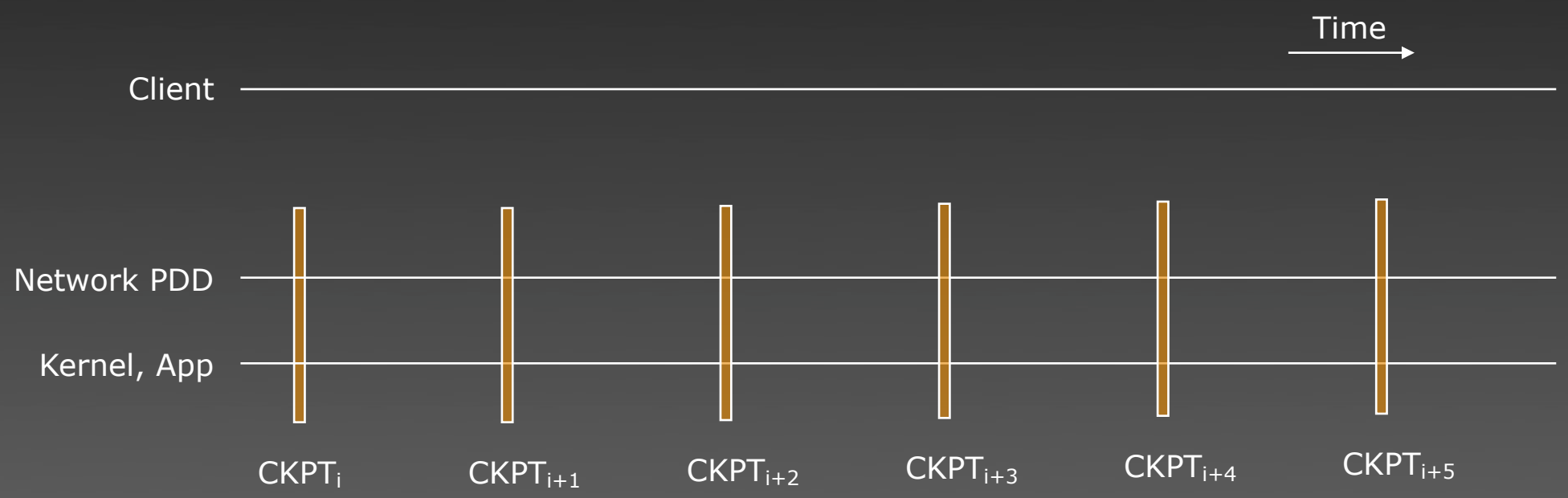
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



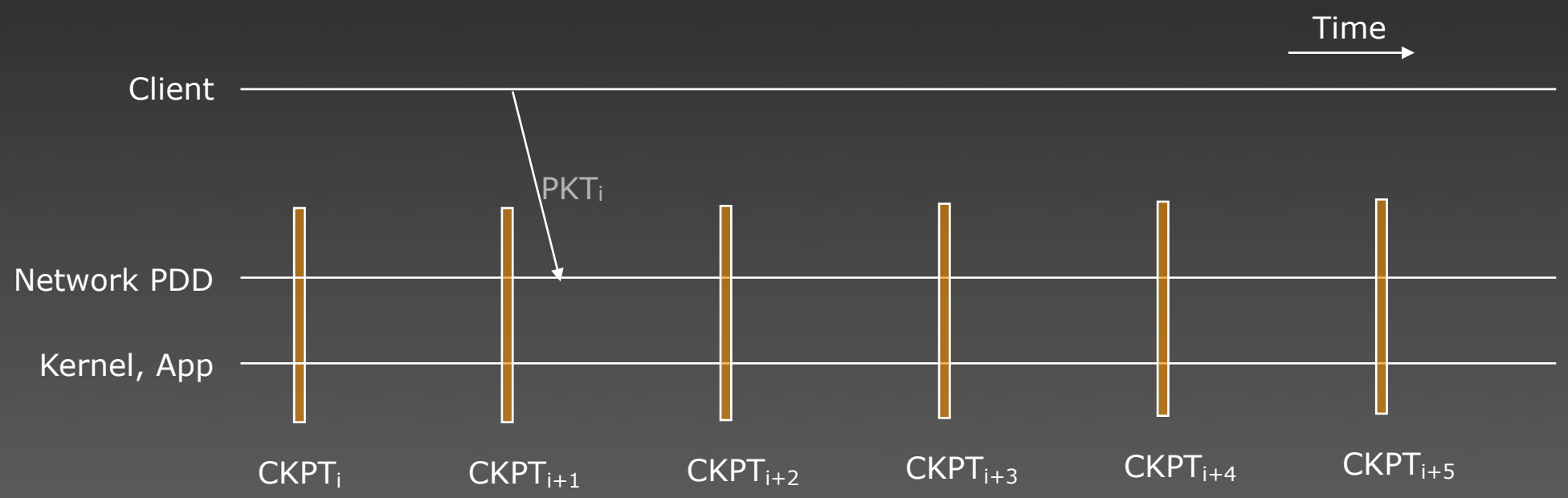
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



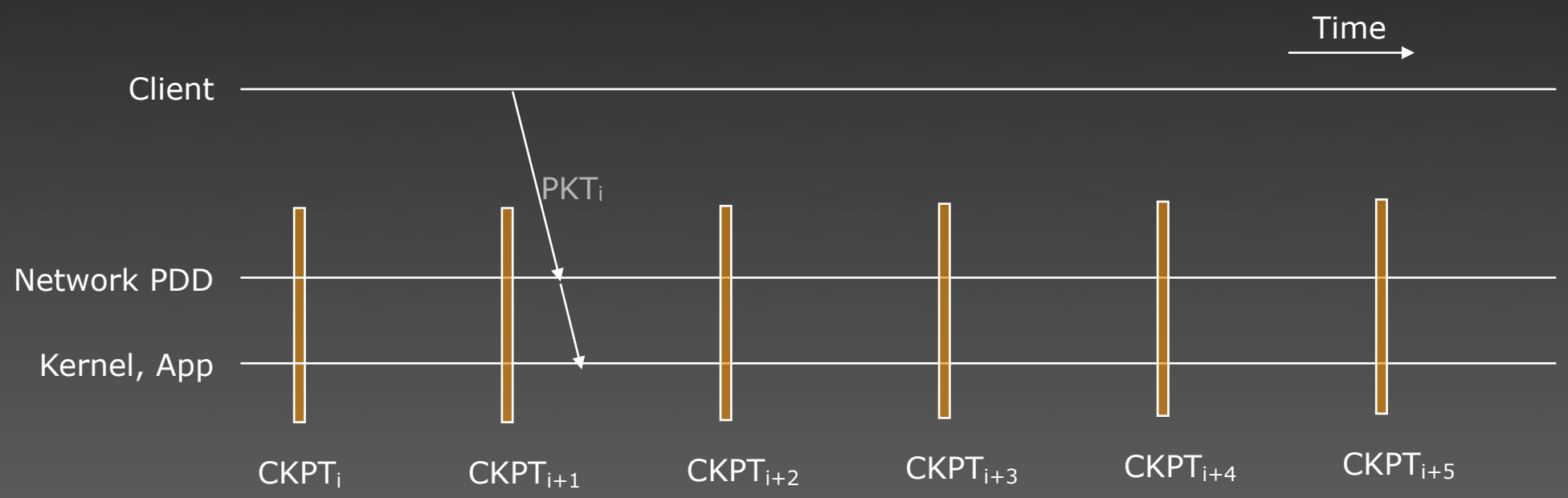
- ReViveI/O leverages the properties of TCP to support network undo/redo

Network PDD and TCP



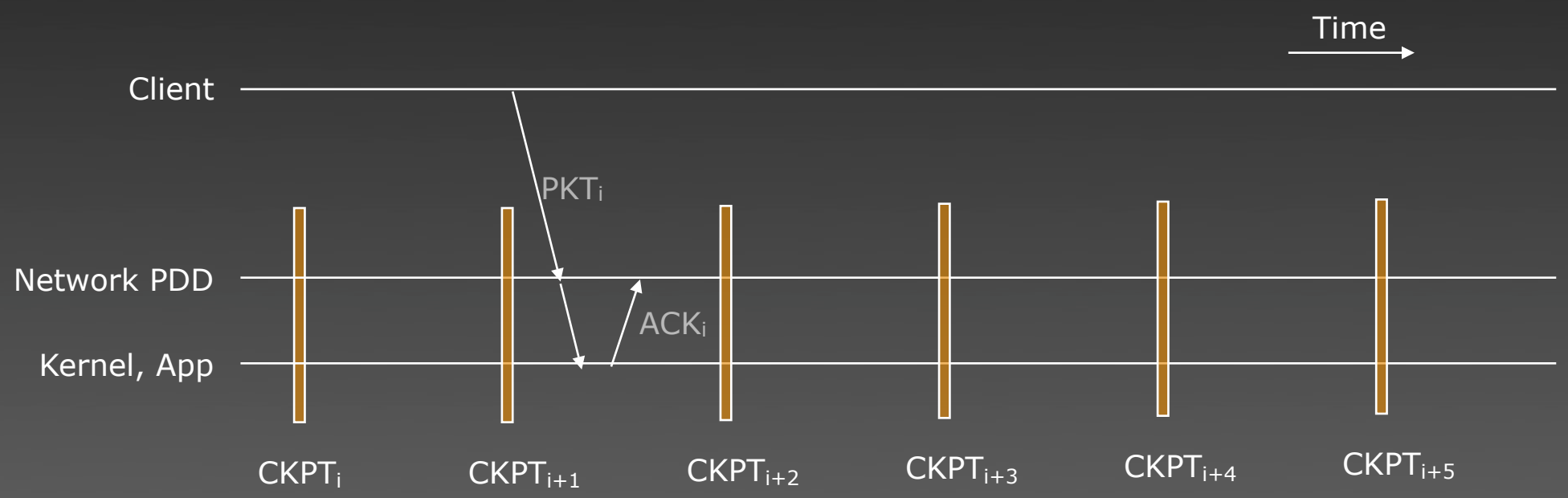
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



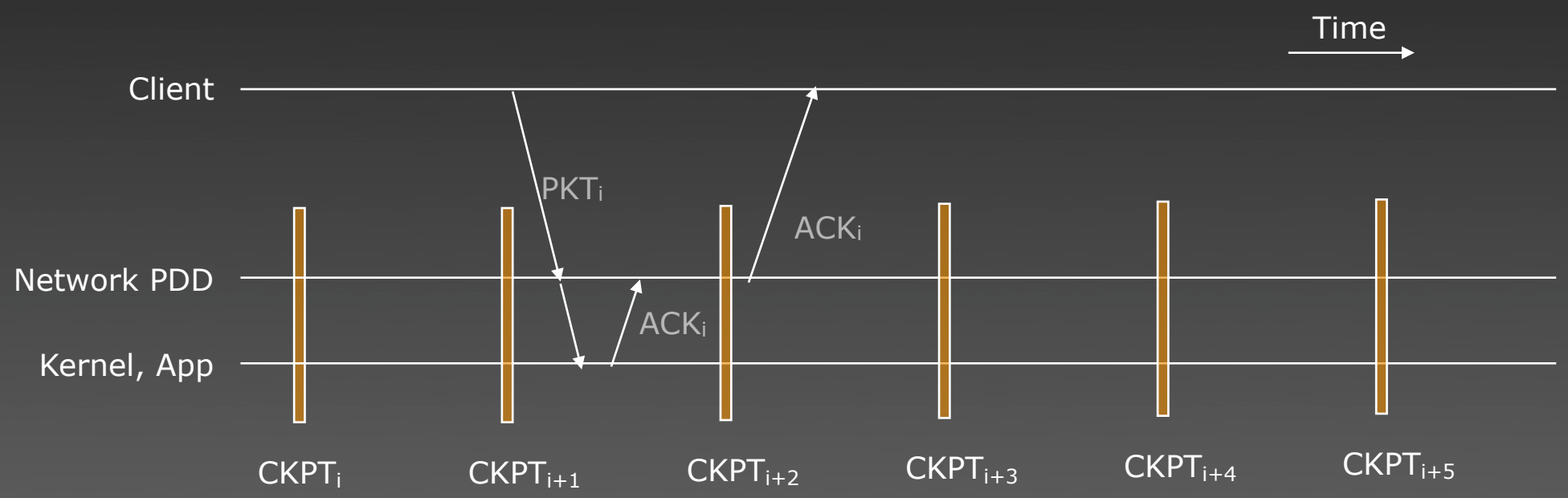
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



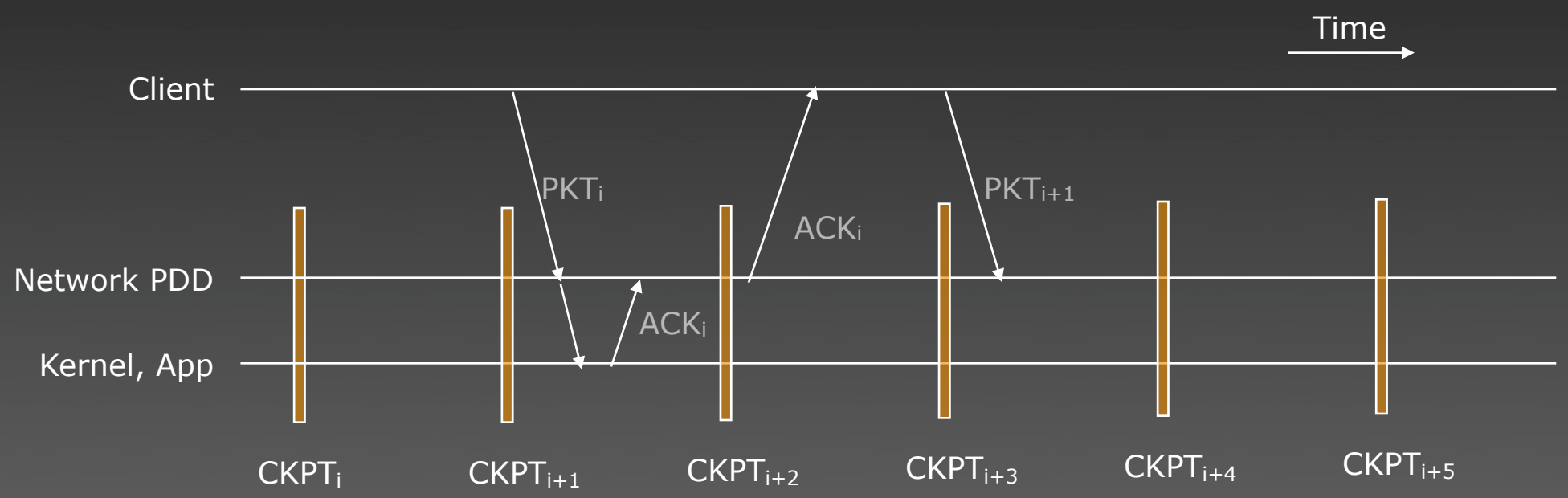
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



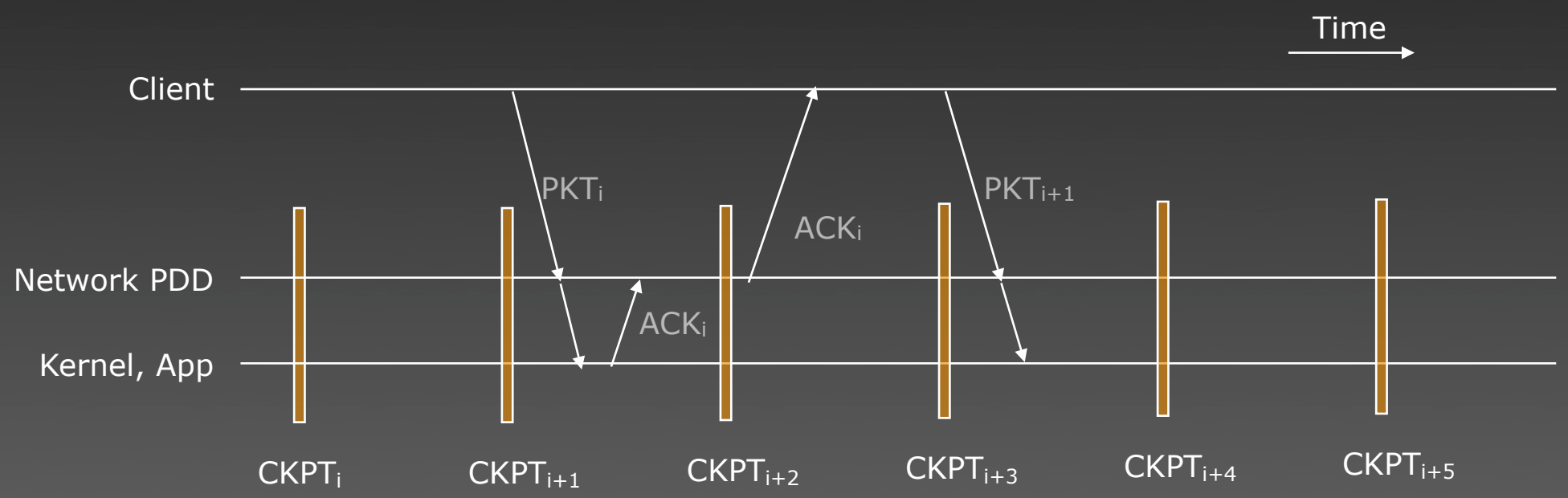
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



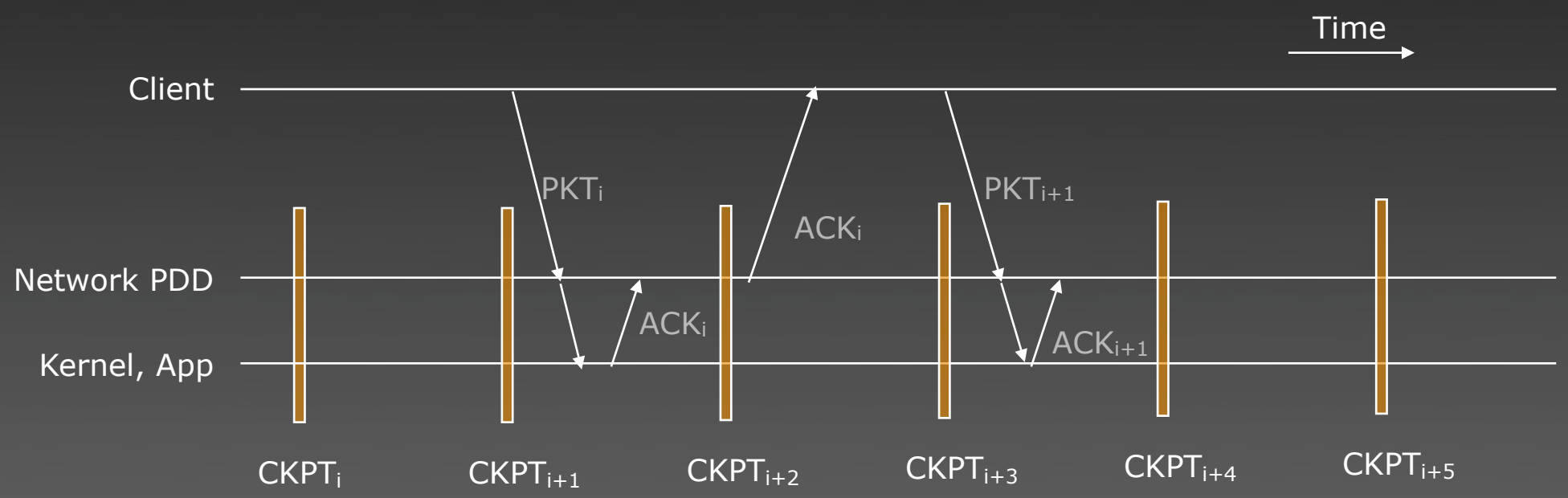
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



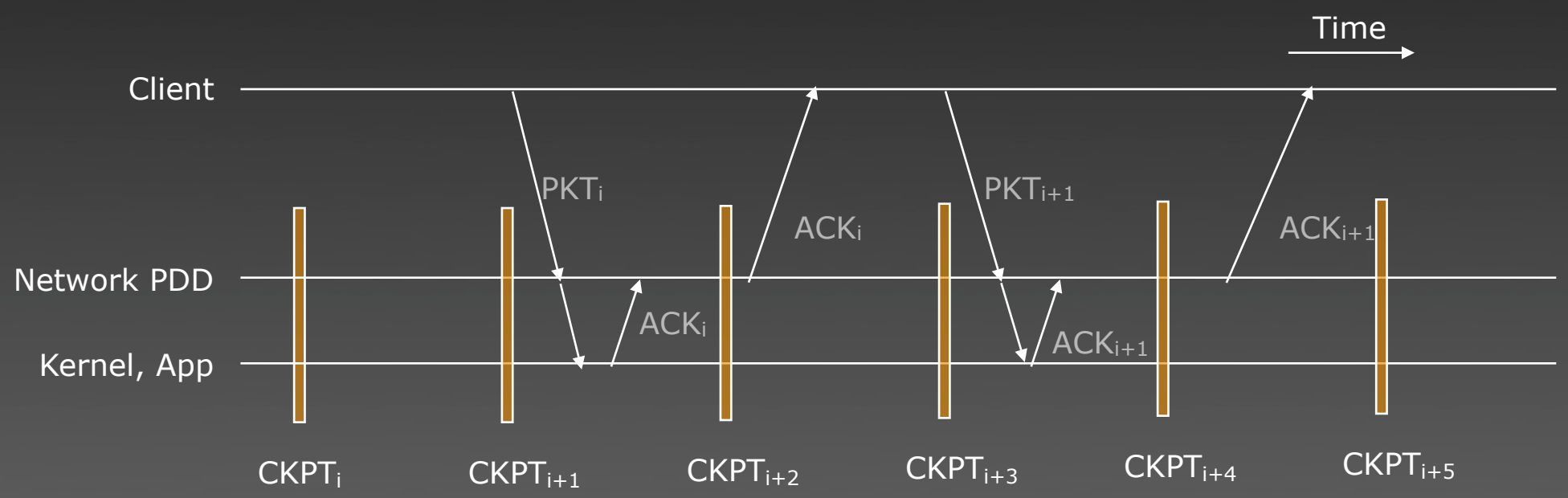
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



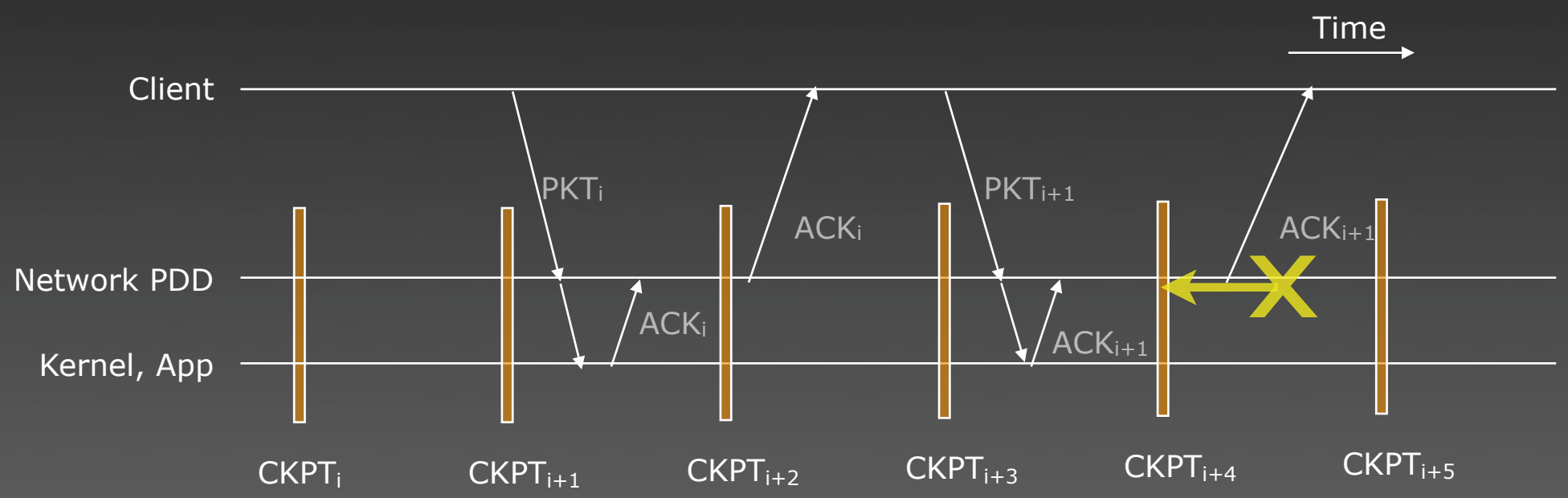
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



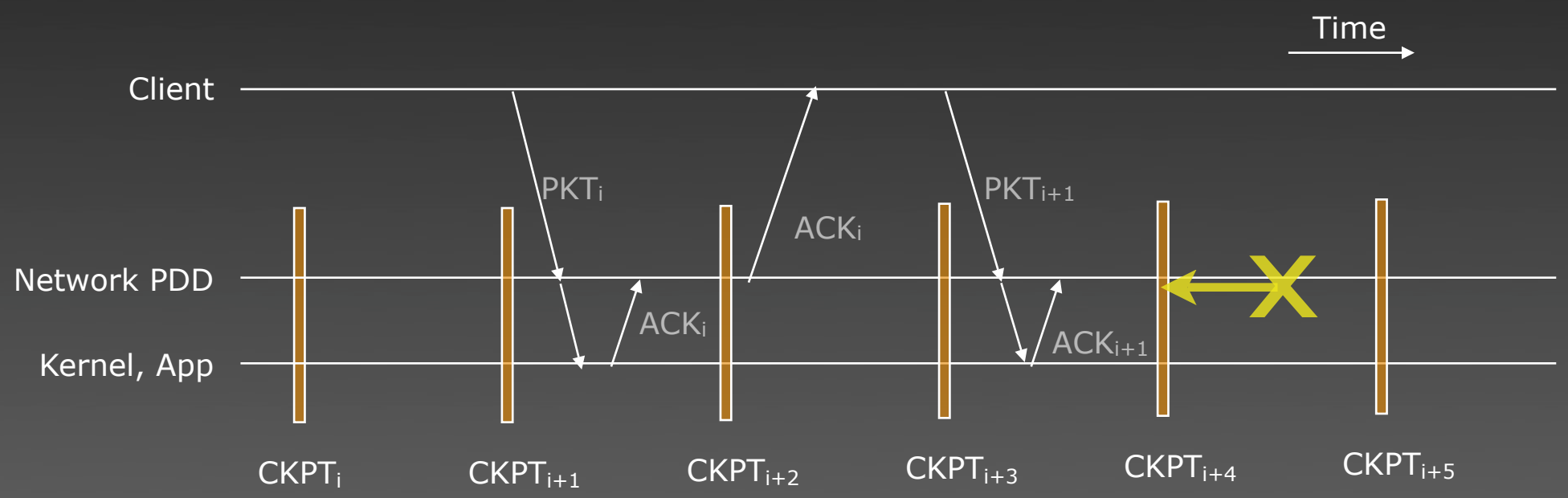
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



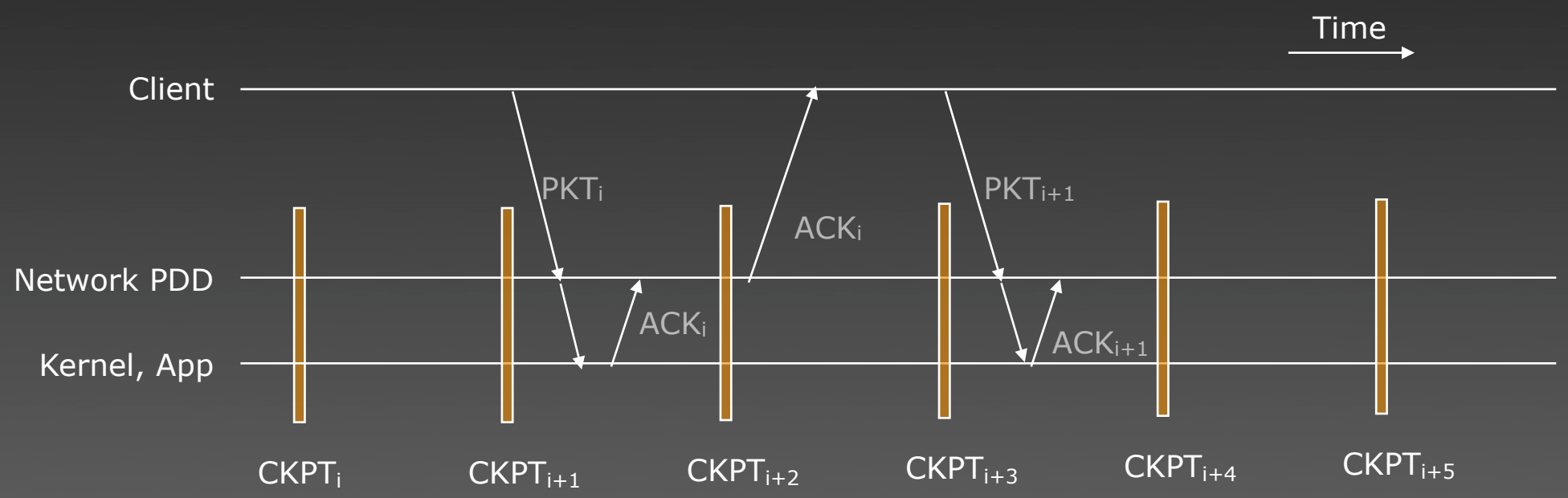
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



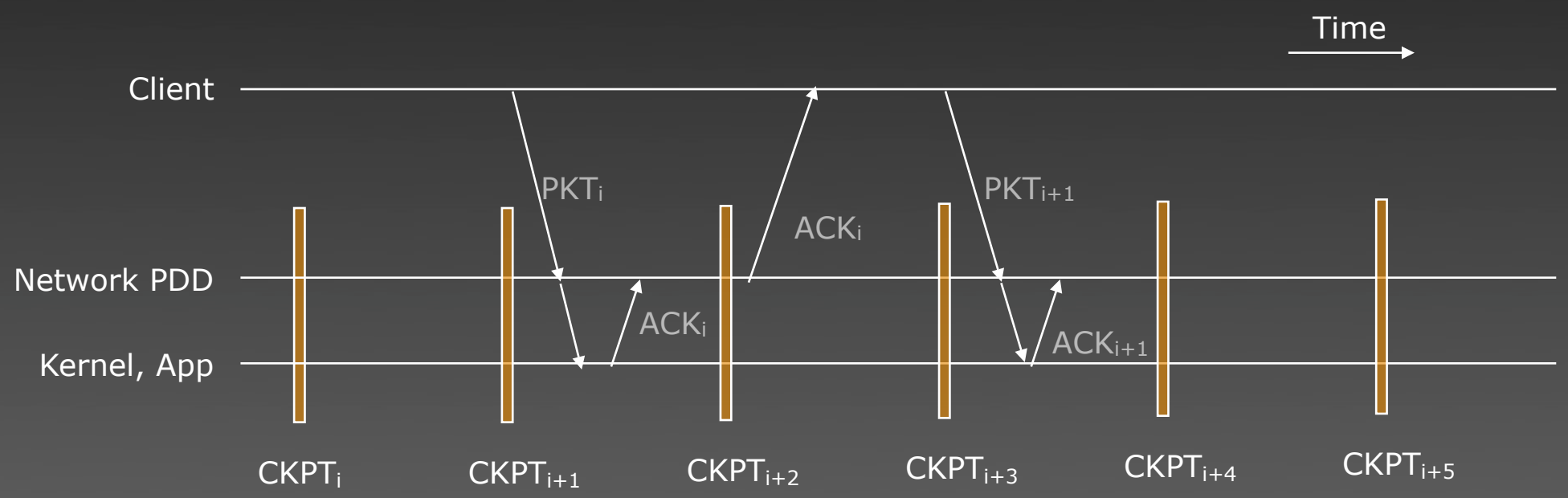
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



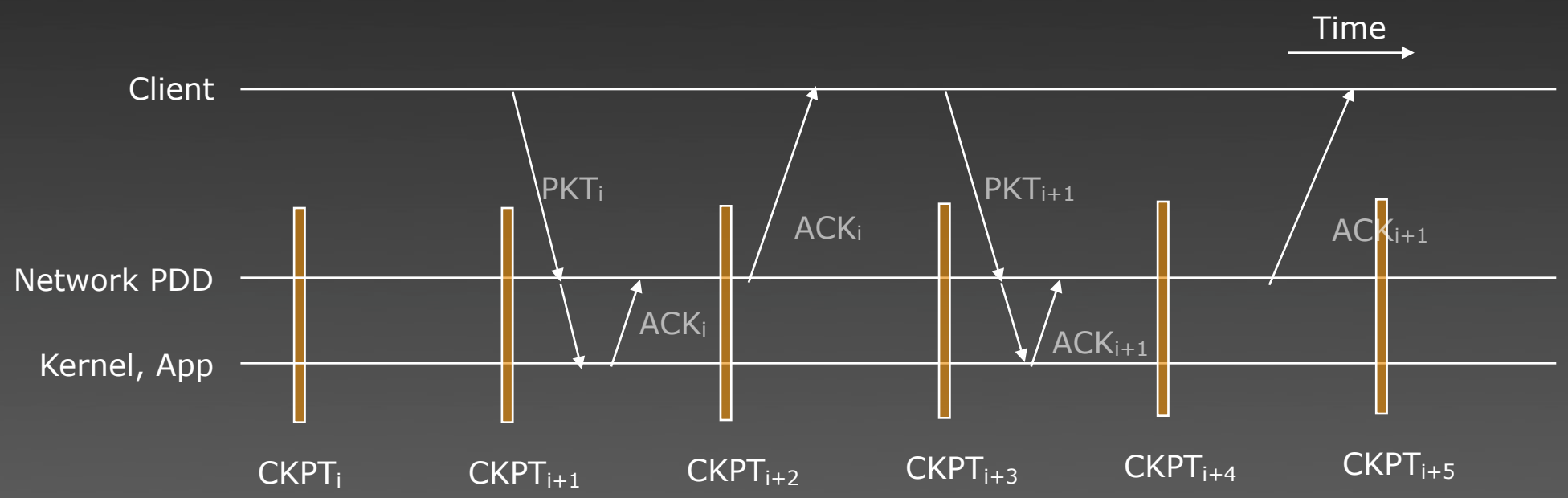
- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits

Network PDD and TCP



- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits
  - TCP eliminates duplicate packages

Network PDD and TCP



- ReViveI/O leverages the properties of TCP to support network undo/redo
  - TCP avoids saving packages for inputs: the sender timeouts and retransmits
  - TCP eliminates duplicate packages

Disk PDD Schemes

Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint

Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint
Stall	Stall the write	Write data to disk

Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint
Stall	Stall the write	Write data to disk

Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint
Stall	Stall the write	Write data to disk
Logging	Copy old data elsewhere in disk Write data in place	Delete pointer to old data

Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint
Stall	Stall the write	Write data to disk
Logging	Copy old data elsewhere in disk Write data in place	Delete pointer to old data
Buffering	Write data to disk buffer and mem	Copy data from mem to disk

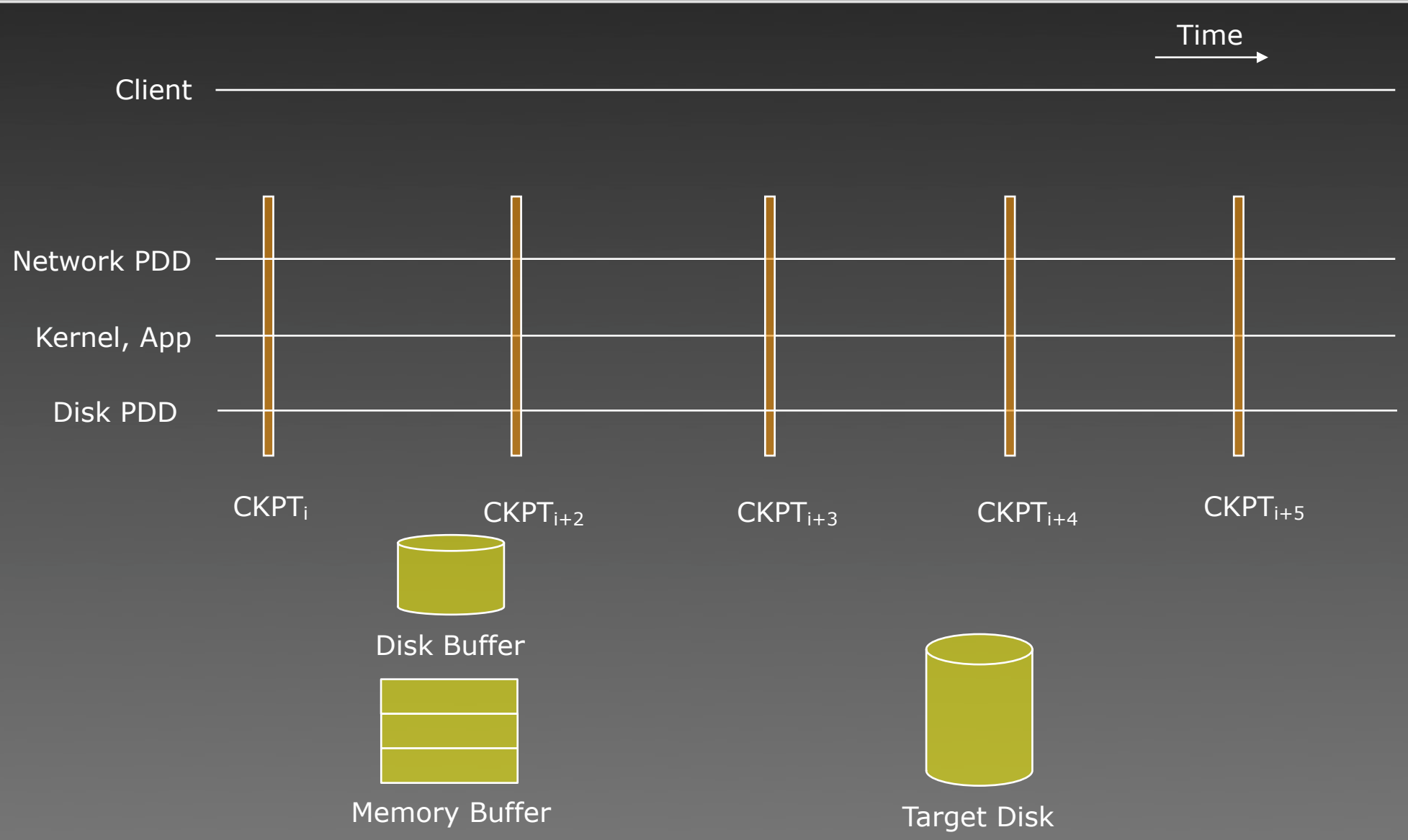
Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint
Stall	Stall the write	Write data to disk
Logging	Copy old data elsewhere in disk Write data in place	Delete pointer to old data
Buffering	Write data to disk buffer and mem	Copy data from mem to disk
Renaming	Write data to renamed disk location Save old logical → physical mapping	Delete old mapping

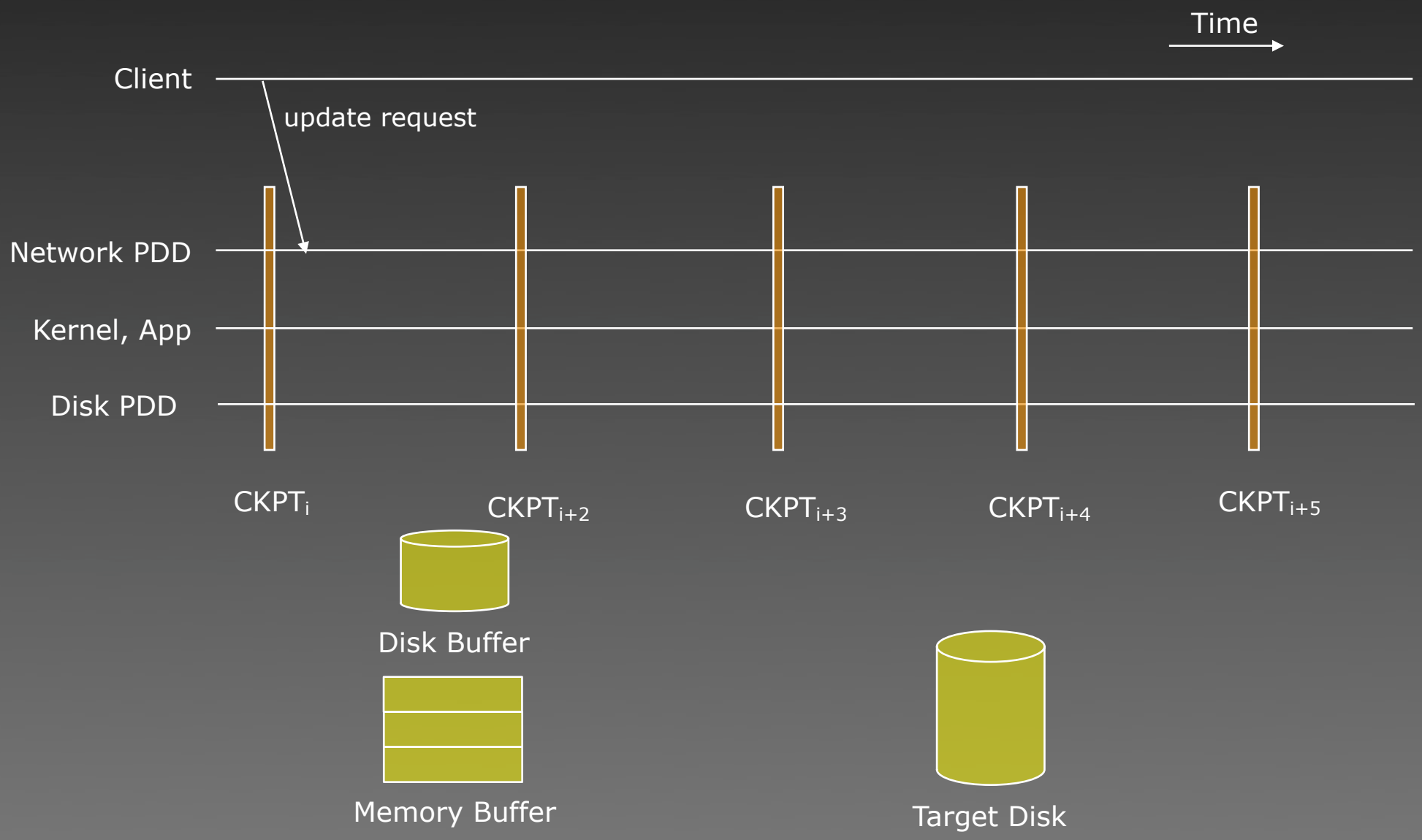
Disk PDD Schemes

Scheme	On I/O write issue	After next checkpoint
Stall	Stall the write	Write data to disk
Logging	Copy old data elsewhere in disk Write data in place	Delete pointer to old data
Buffering	Write data to disk buffer and mem	Copy data from mem to disk
Renaming	Write data to renamed disk location Save old logical → physical mapping	Delete old mapping

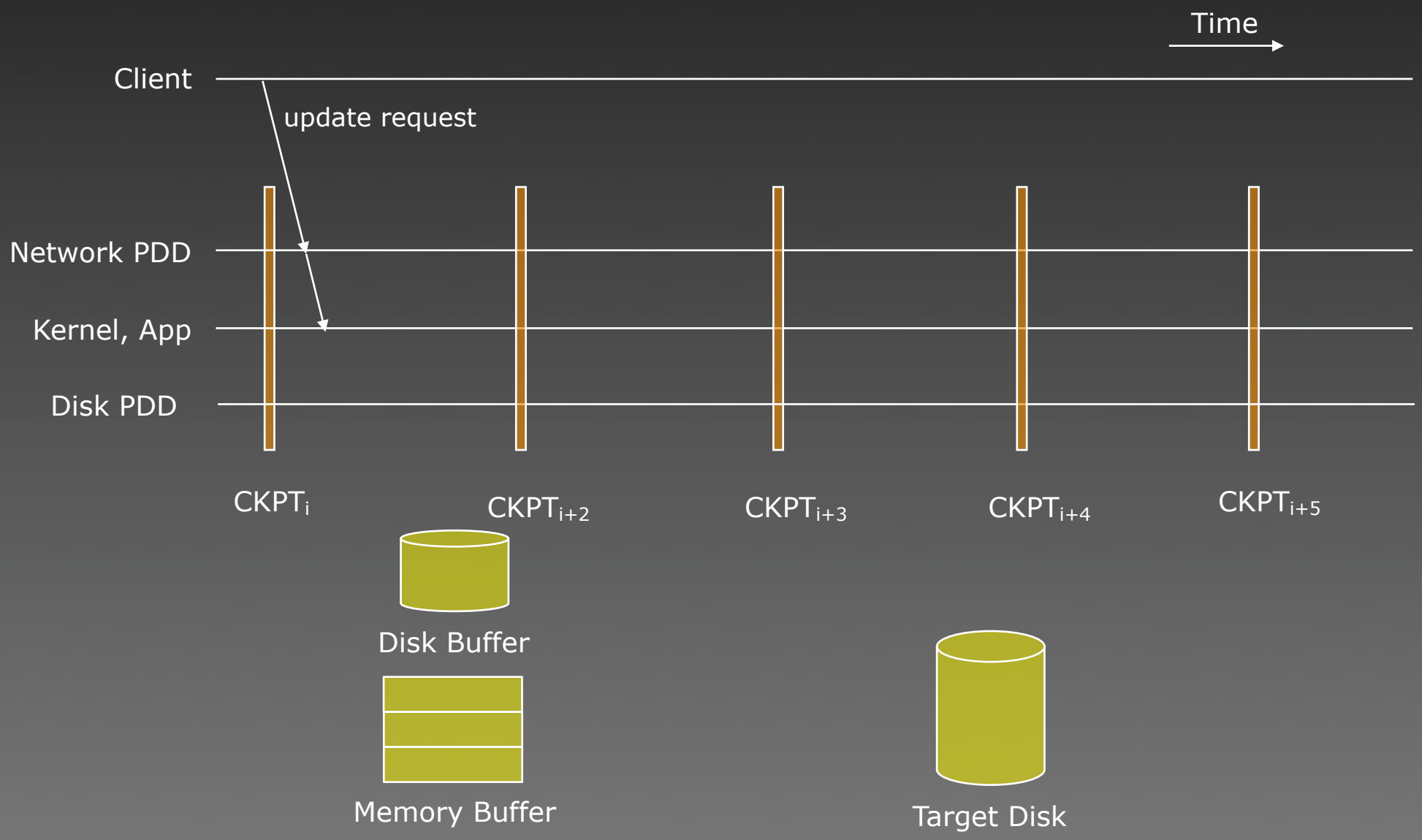
Disk PDD


Time
Client
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk

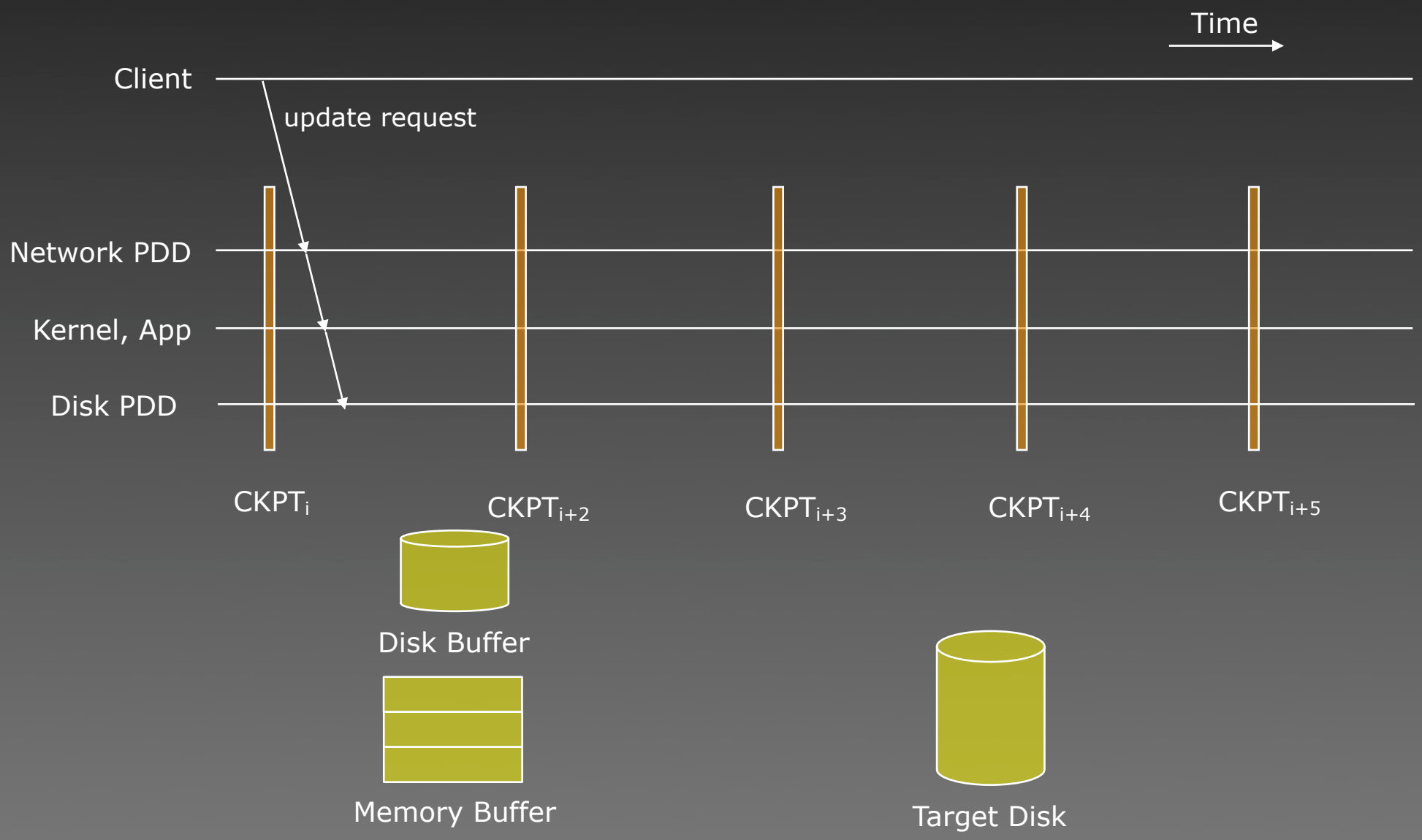
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk

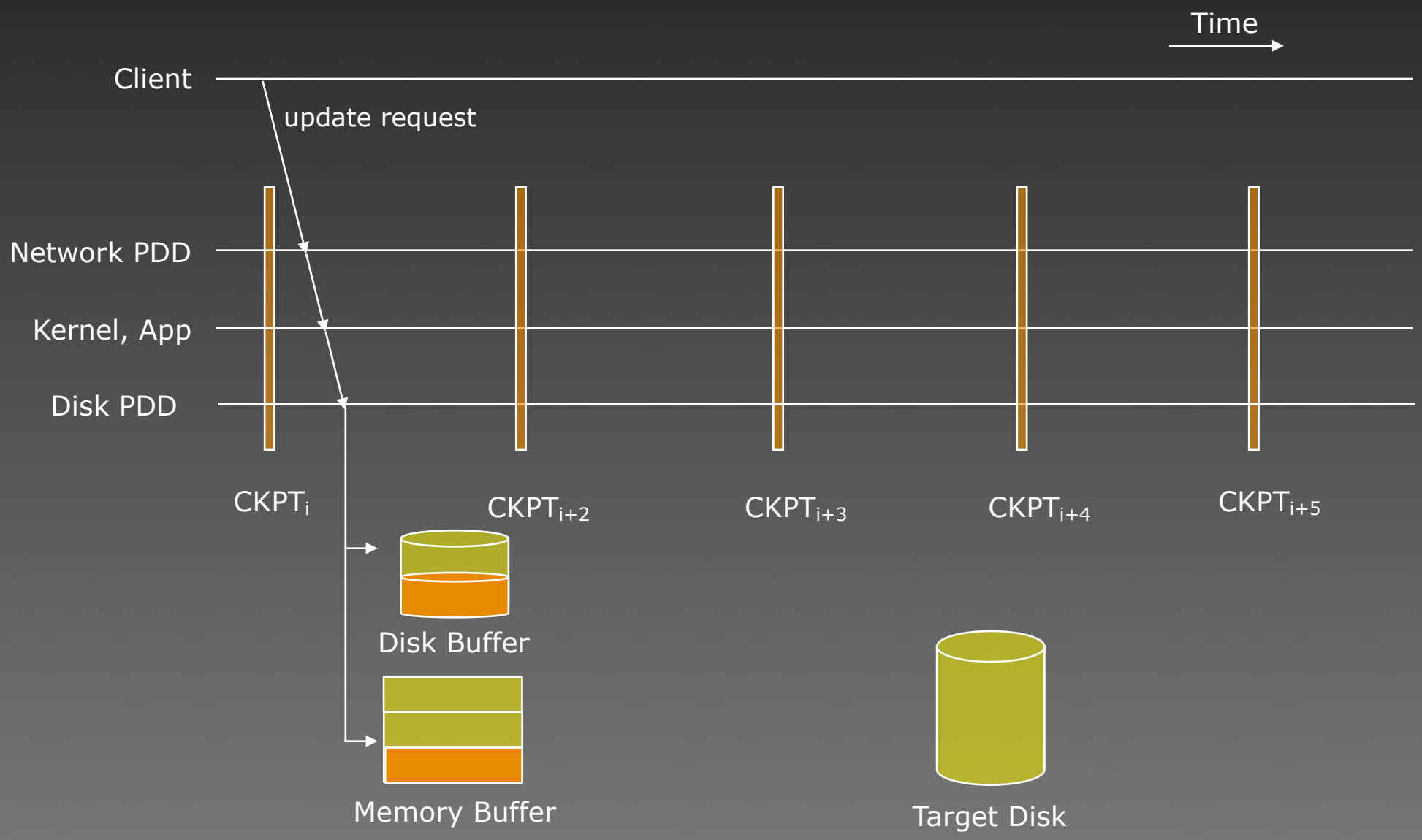
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk

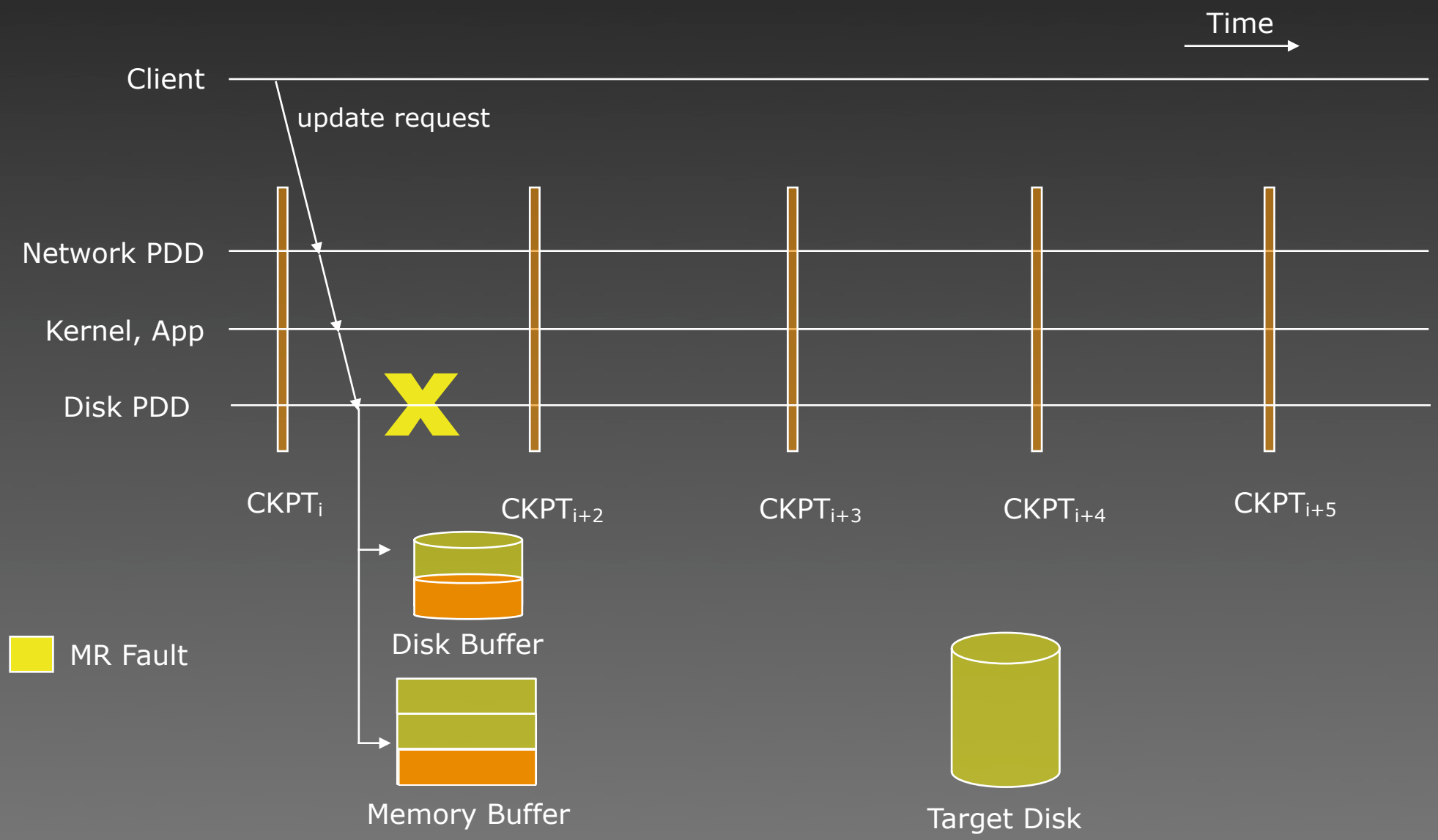
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk

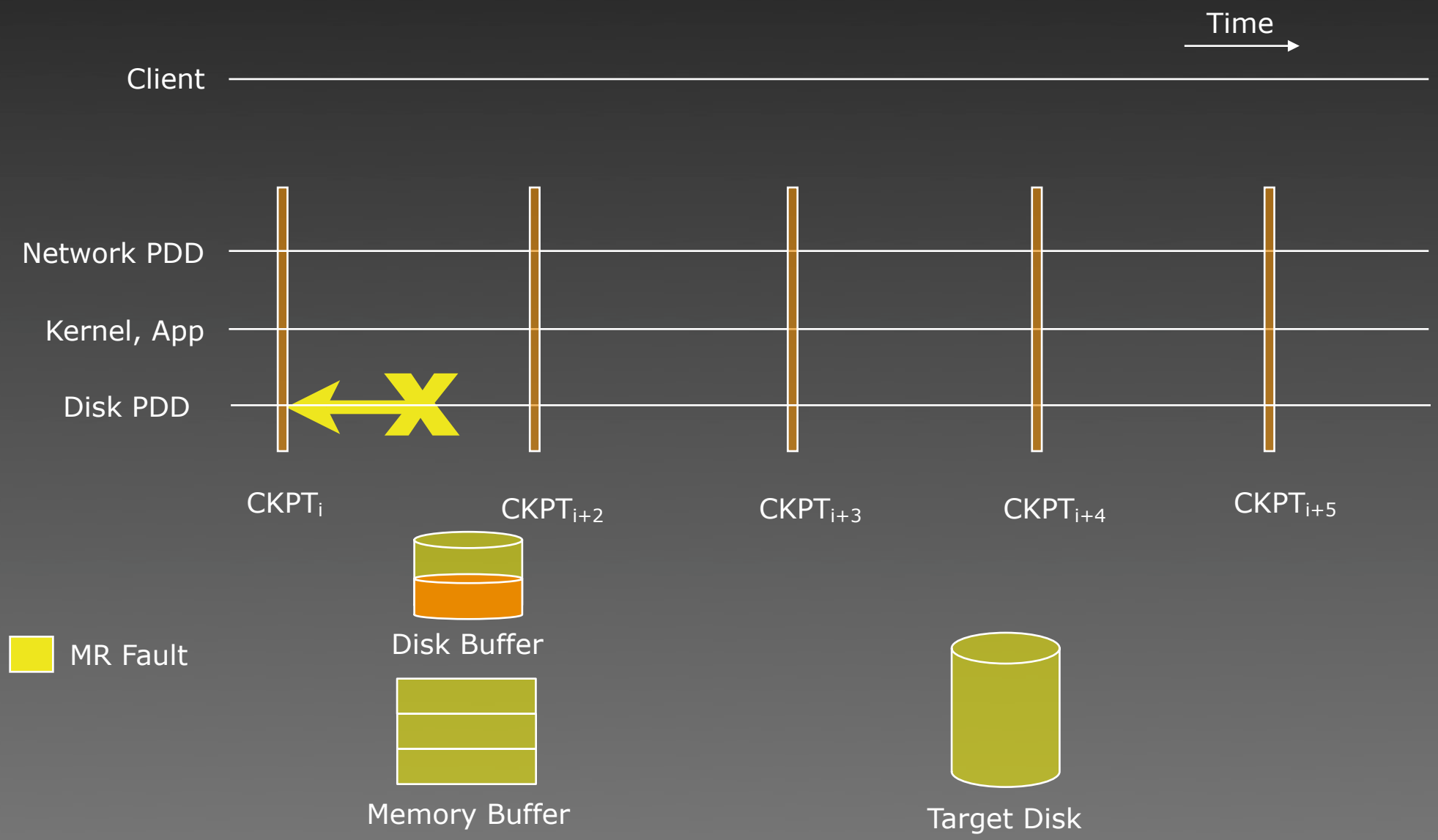
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk

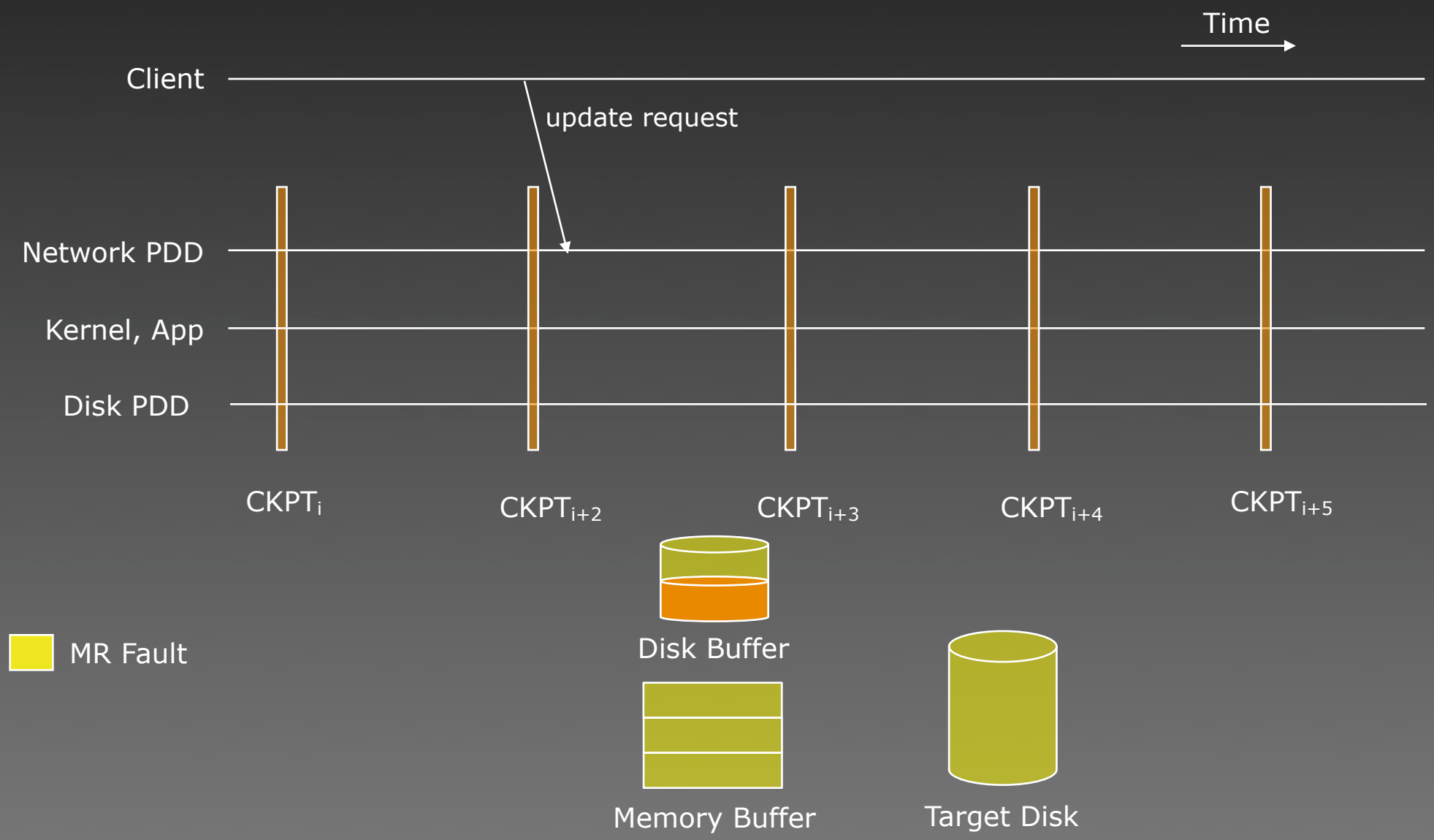
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

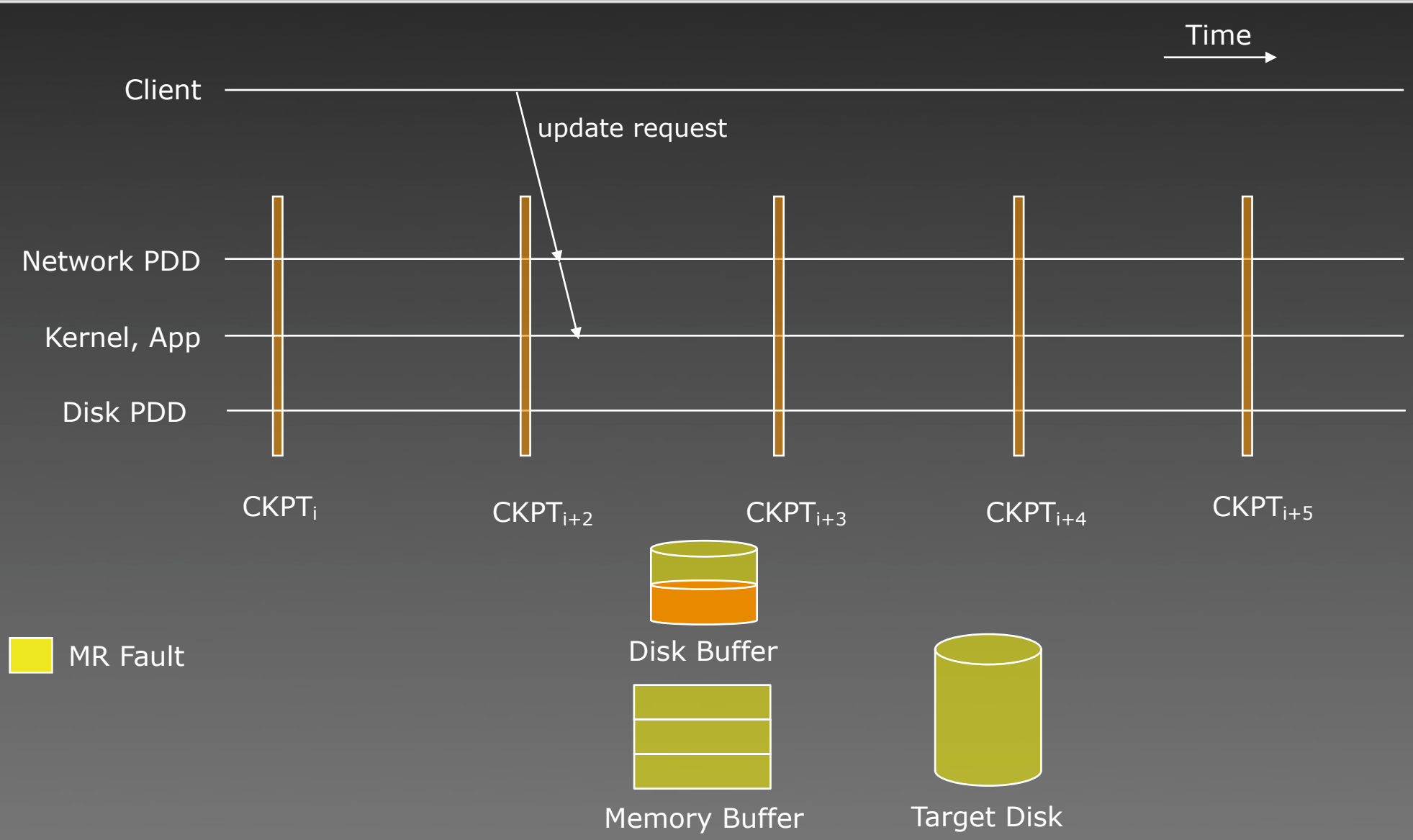
Disk PDD


Time
Client
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

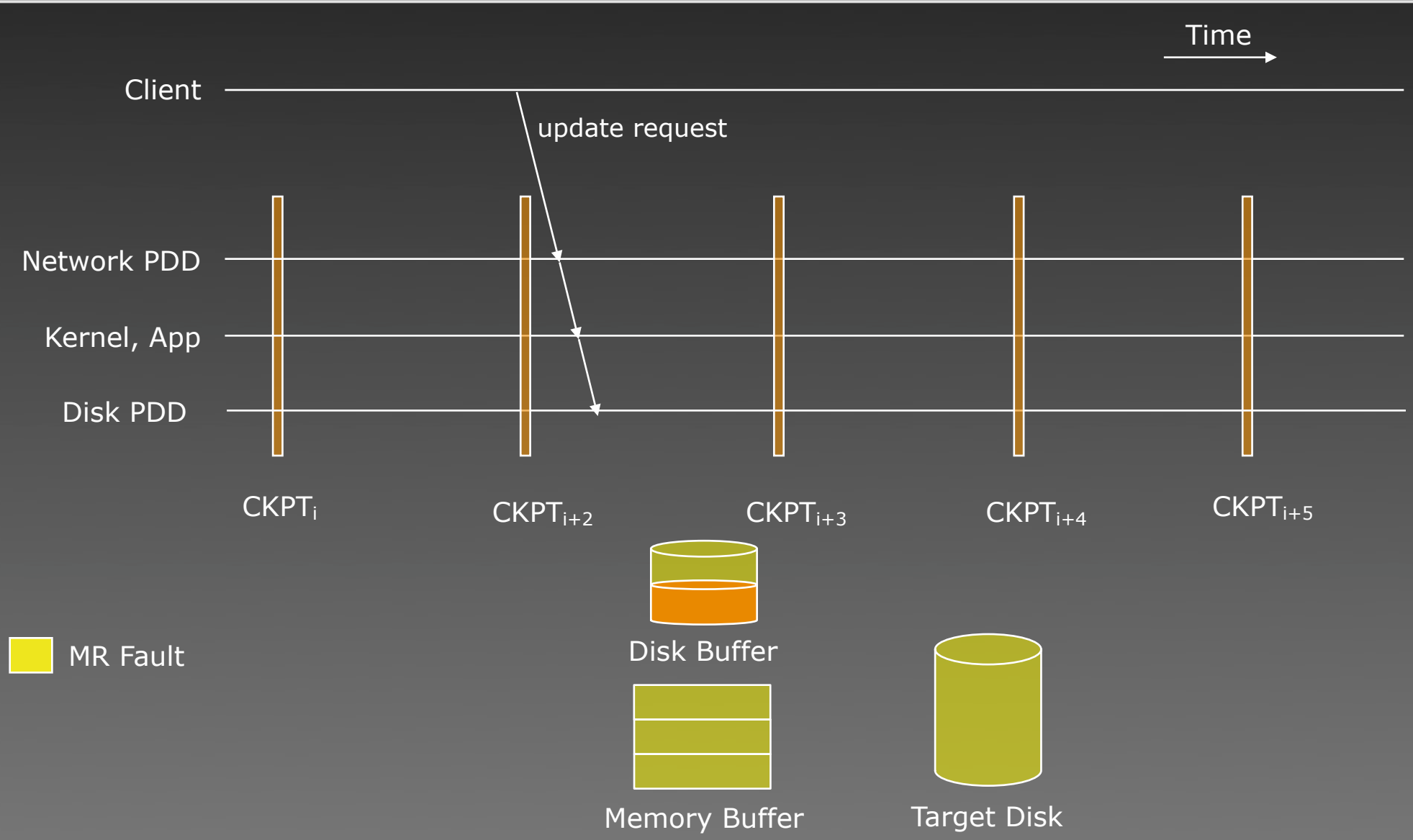
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

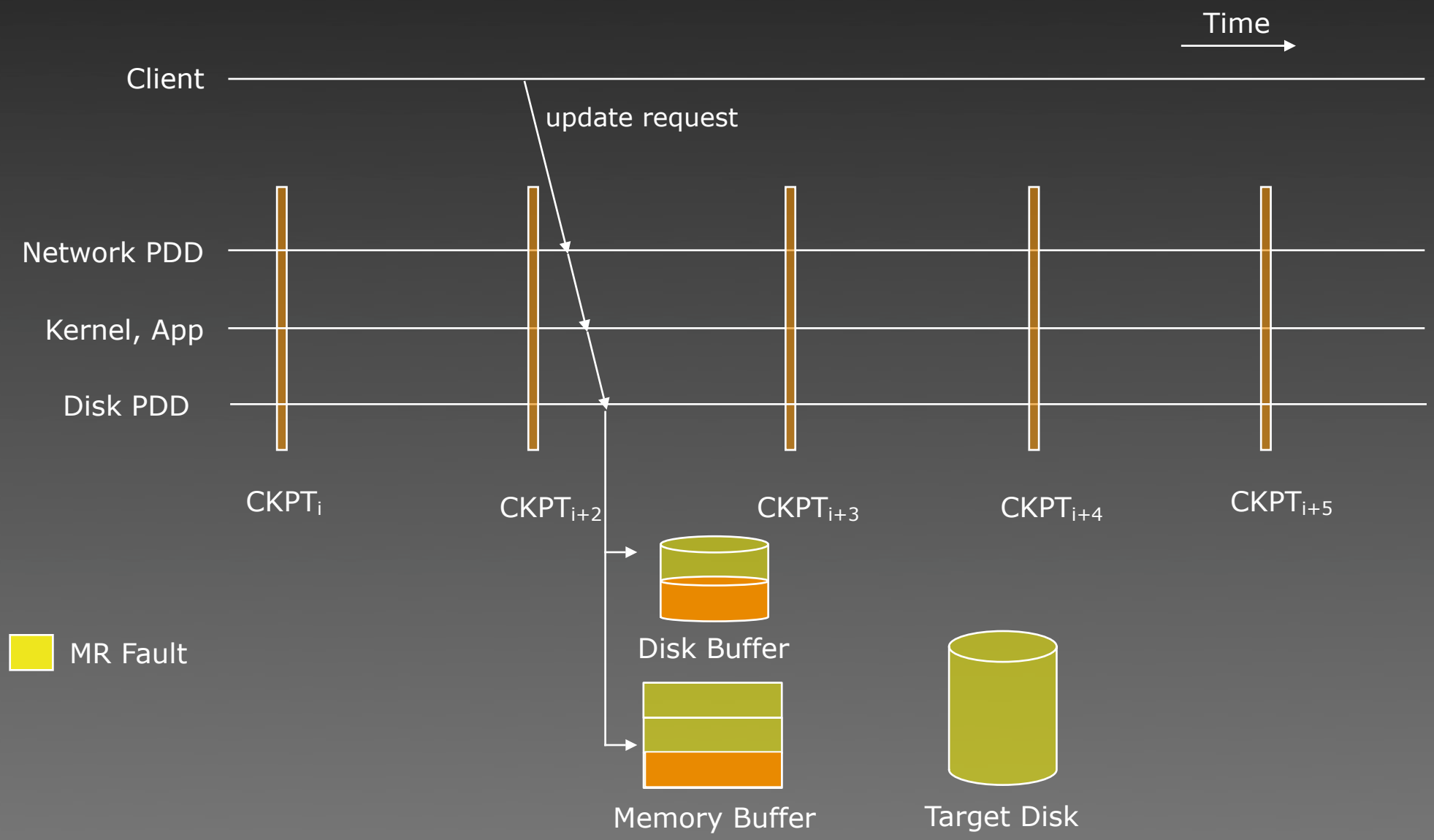
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

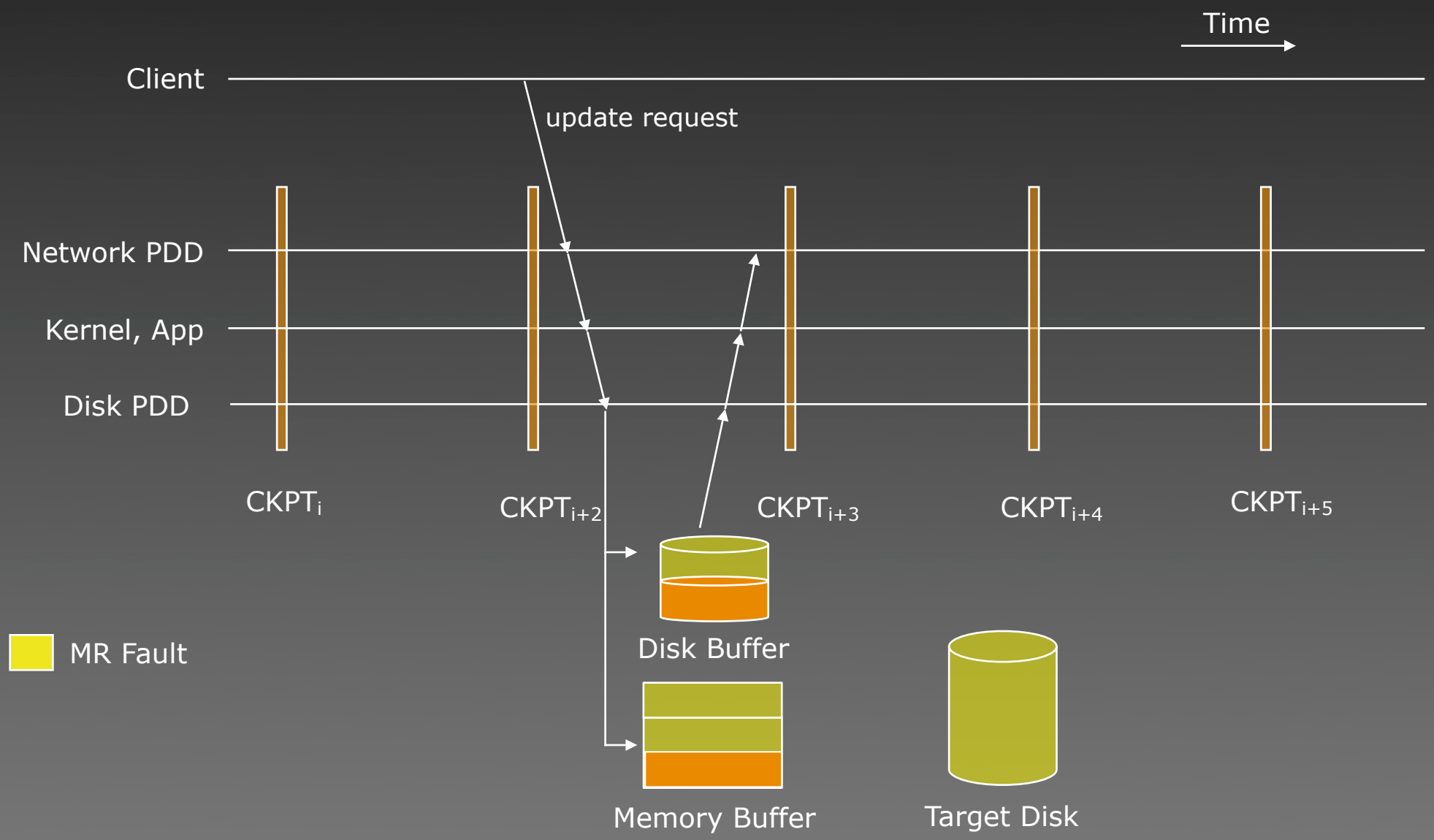
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

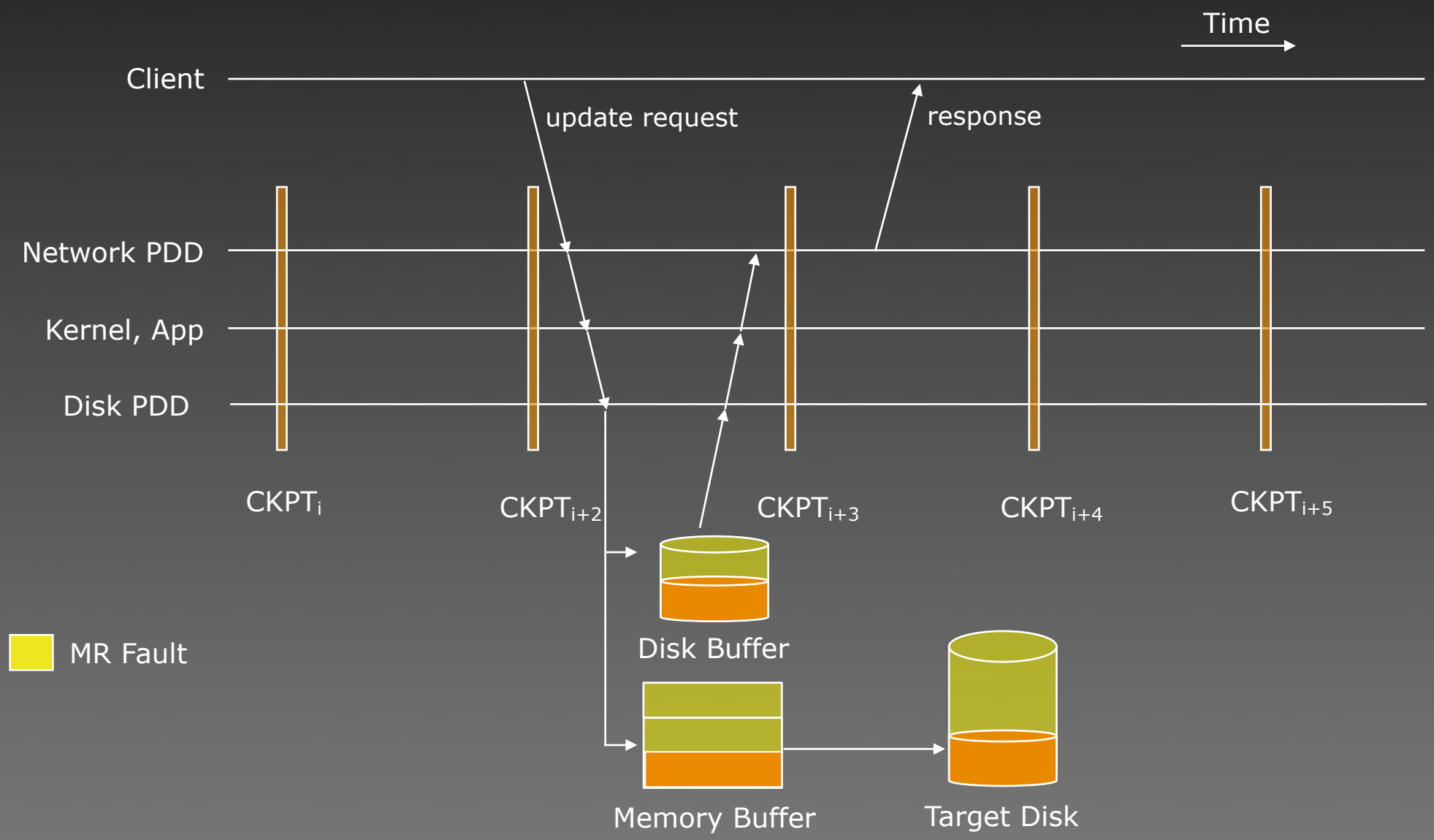
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk
MR Fault

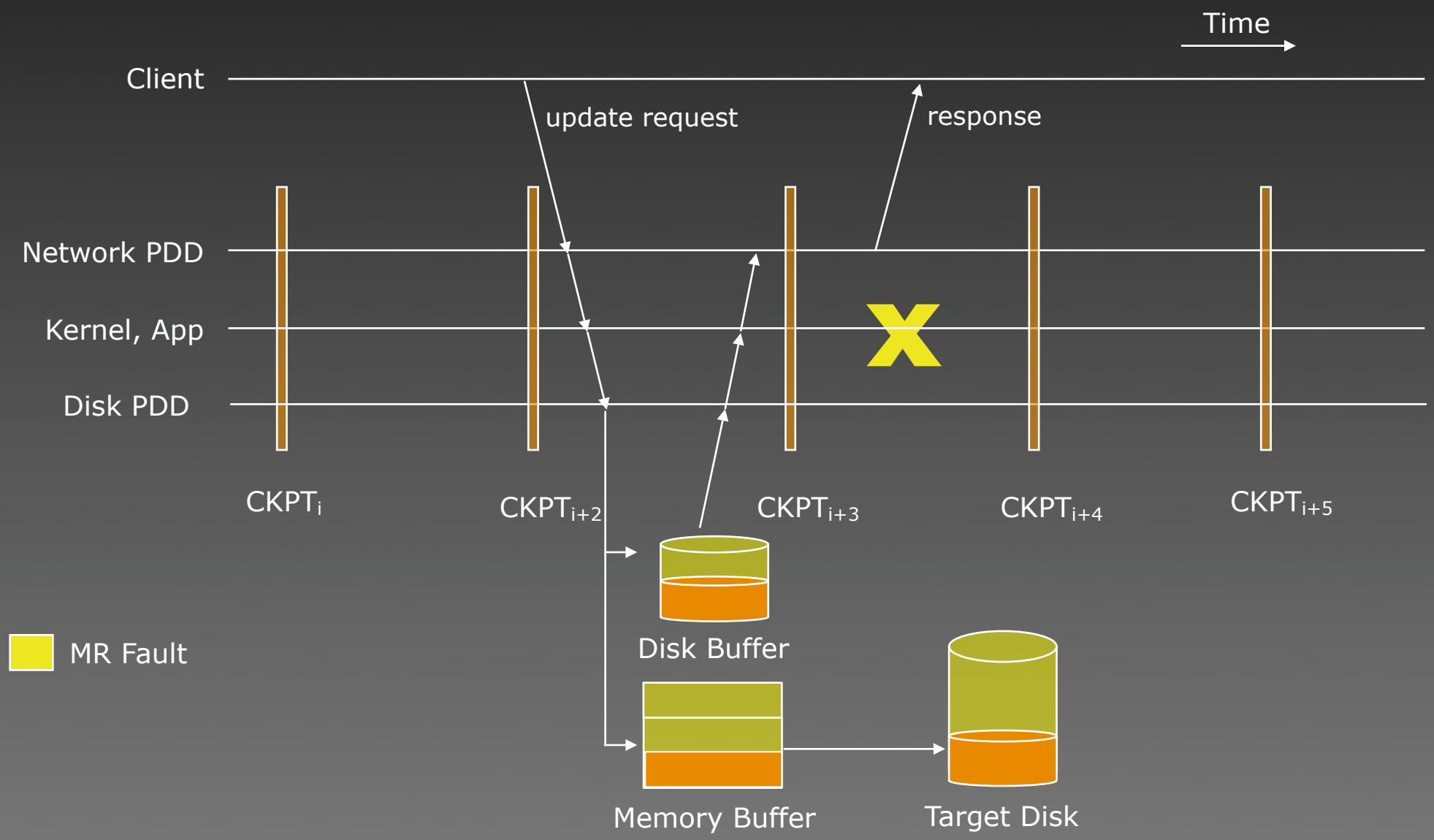
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk
MR Fault

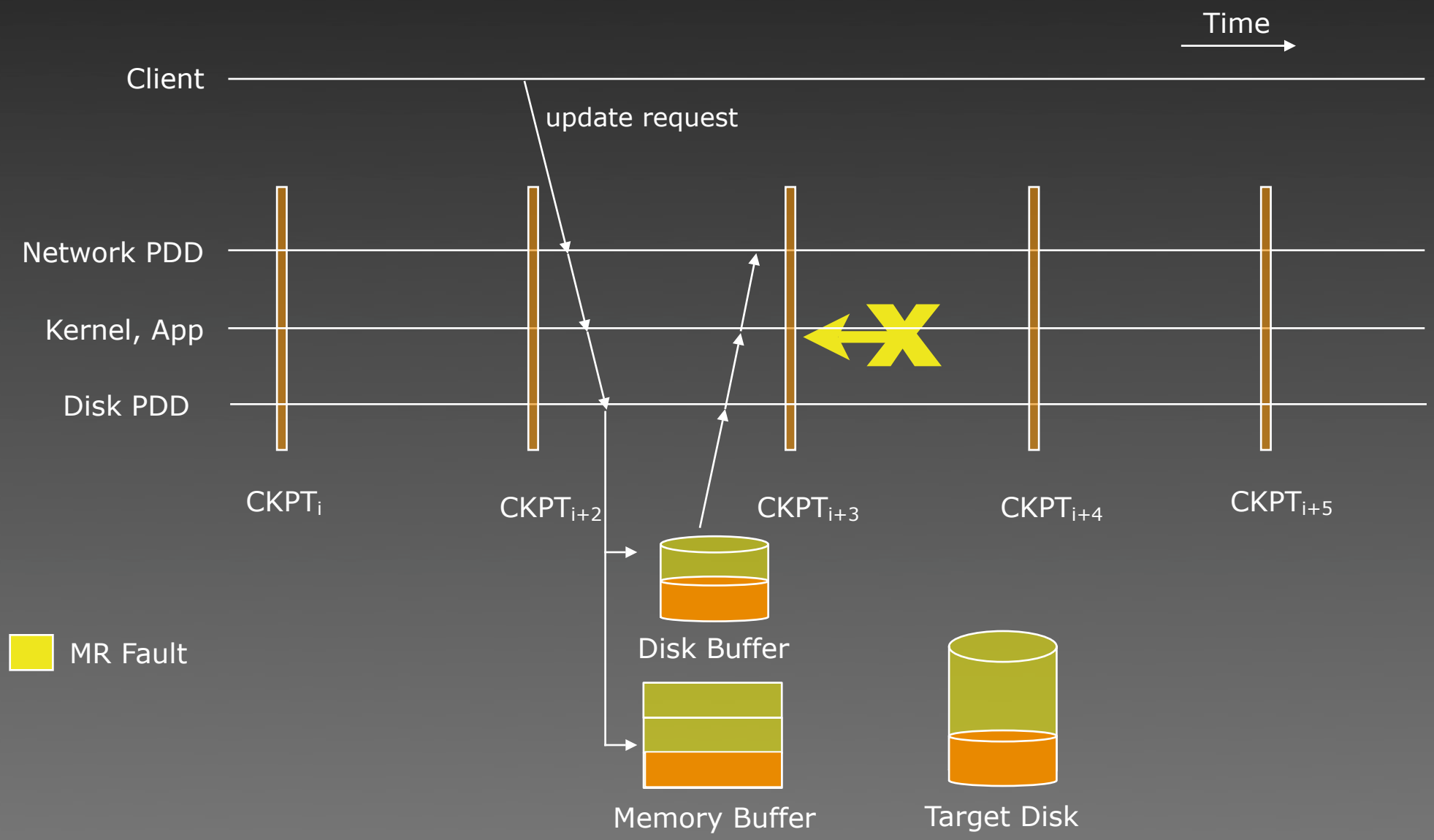
Disk PDD


Time
Client
update request
response
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

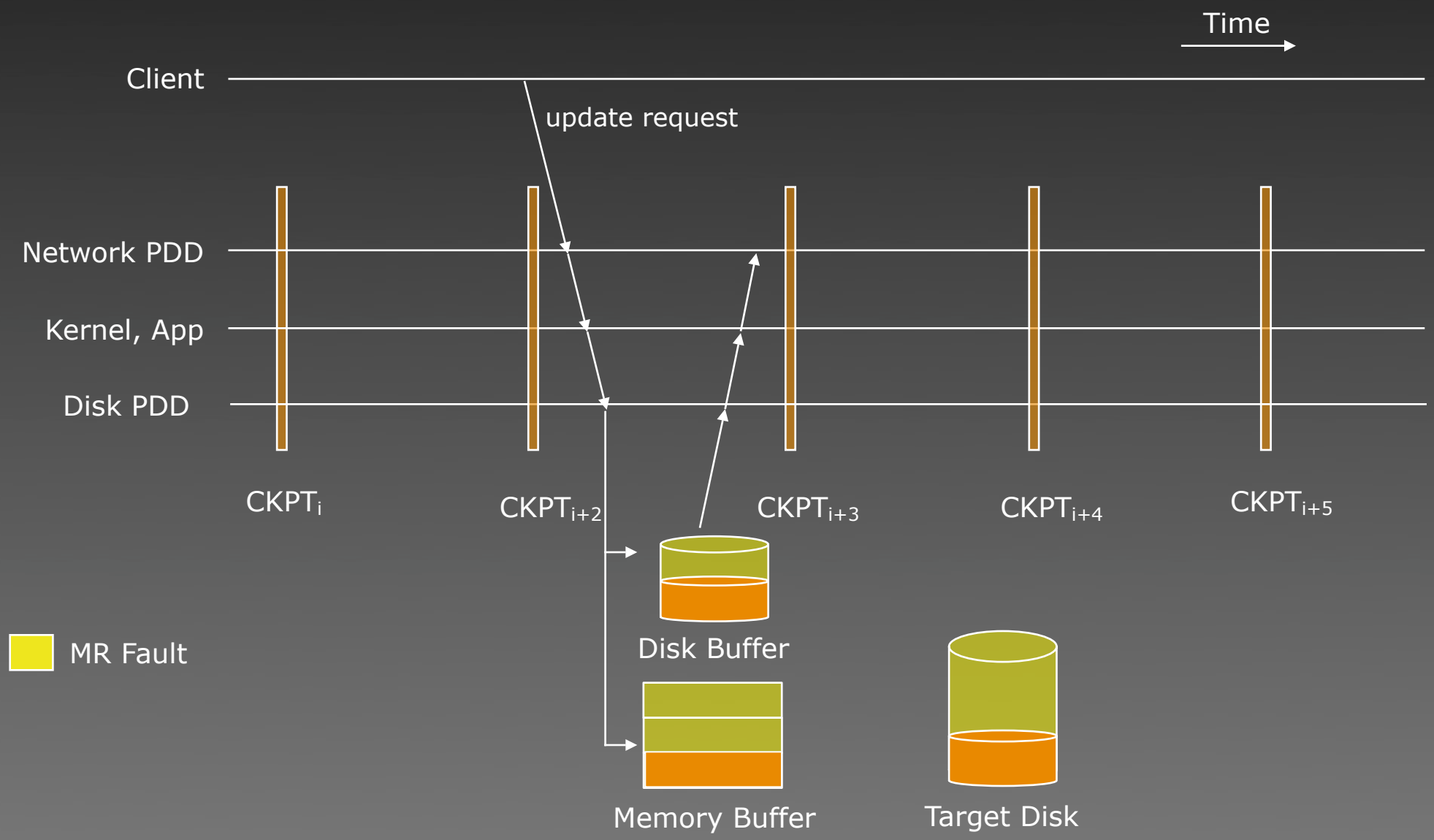
Disk PDD


Time
Client
update request
response
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

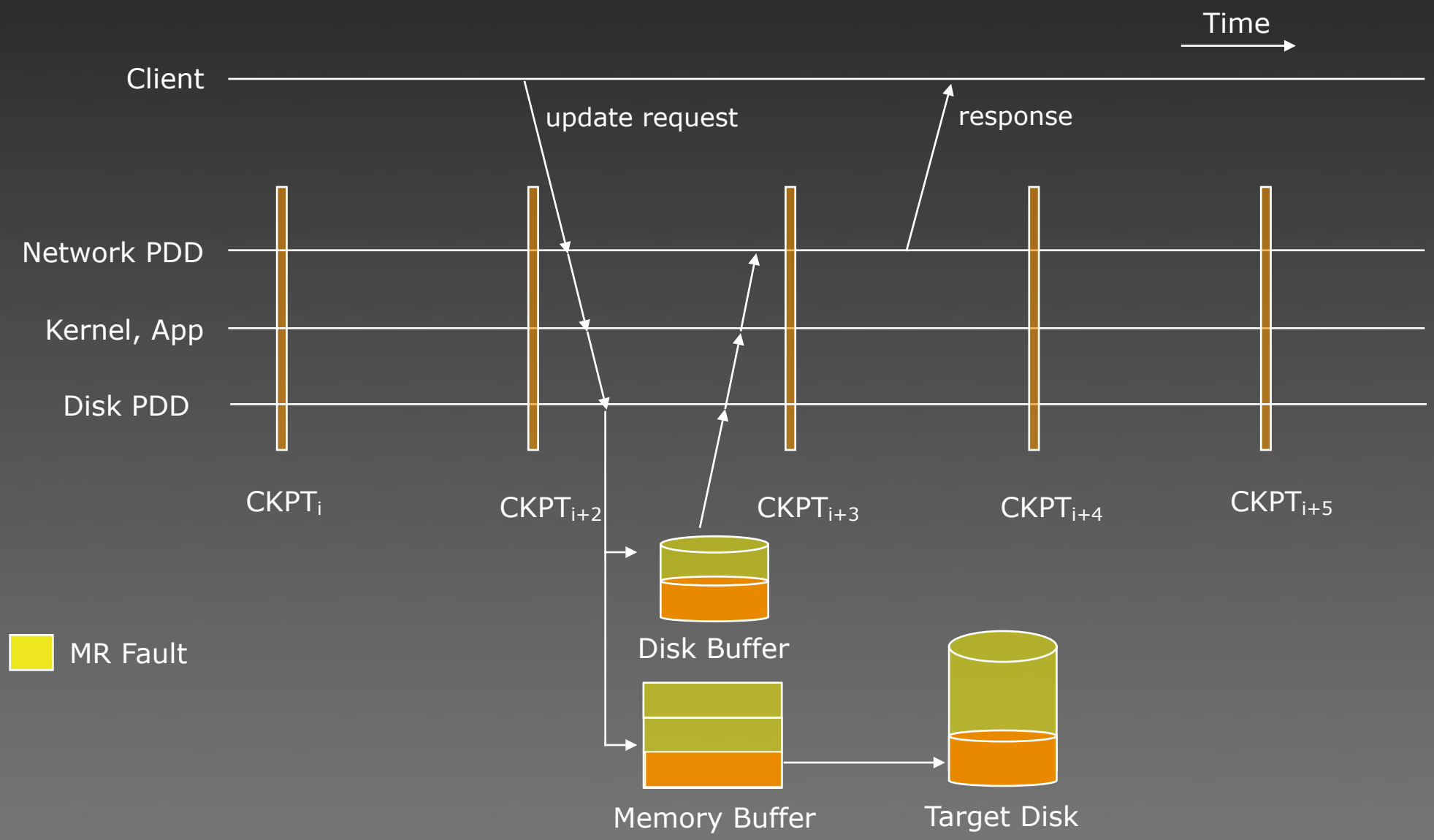
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk
MR Fault

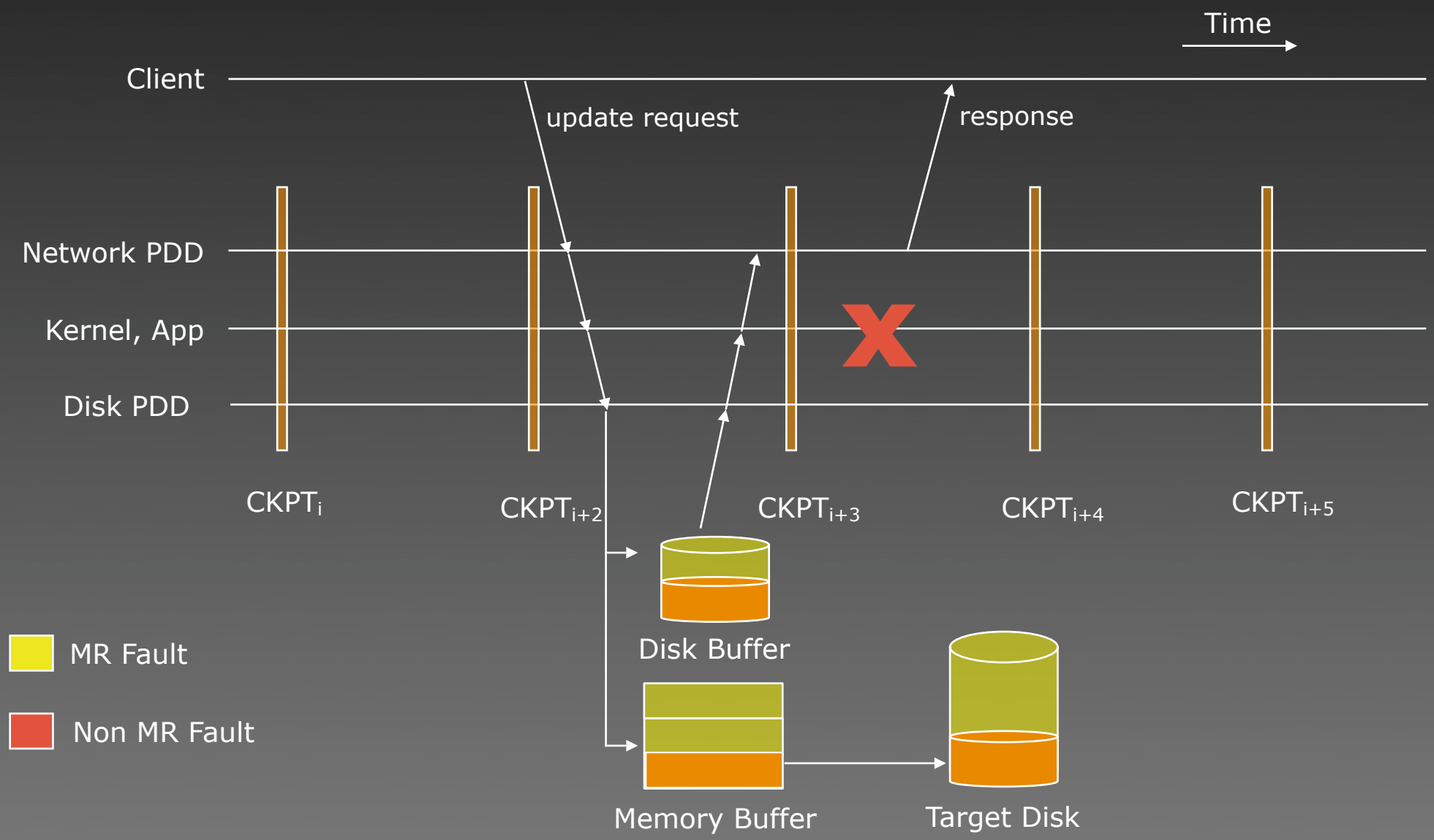
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk
MR Fault

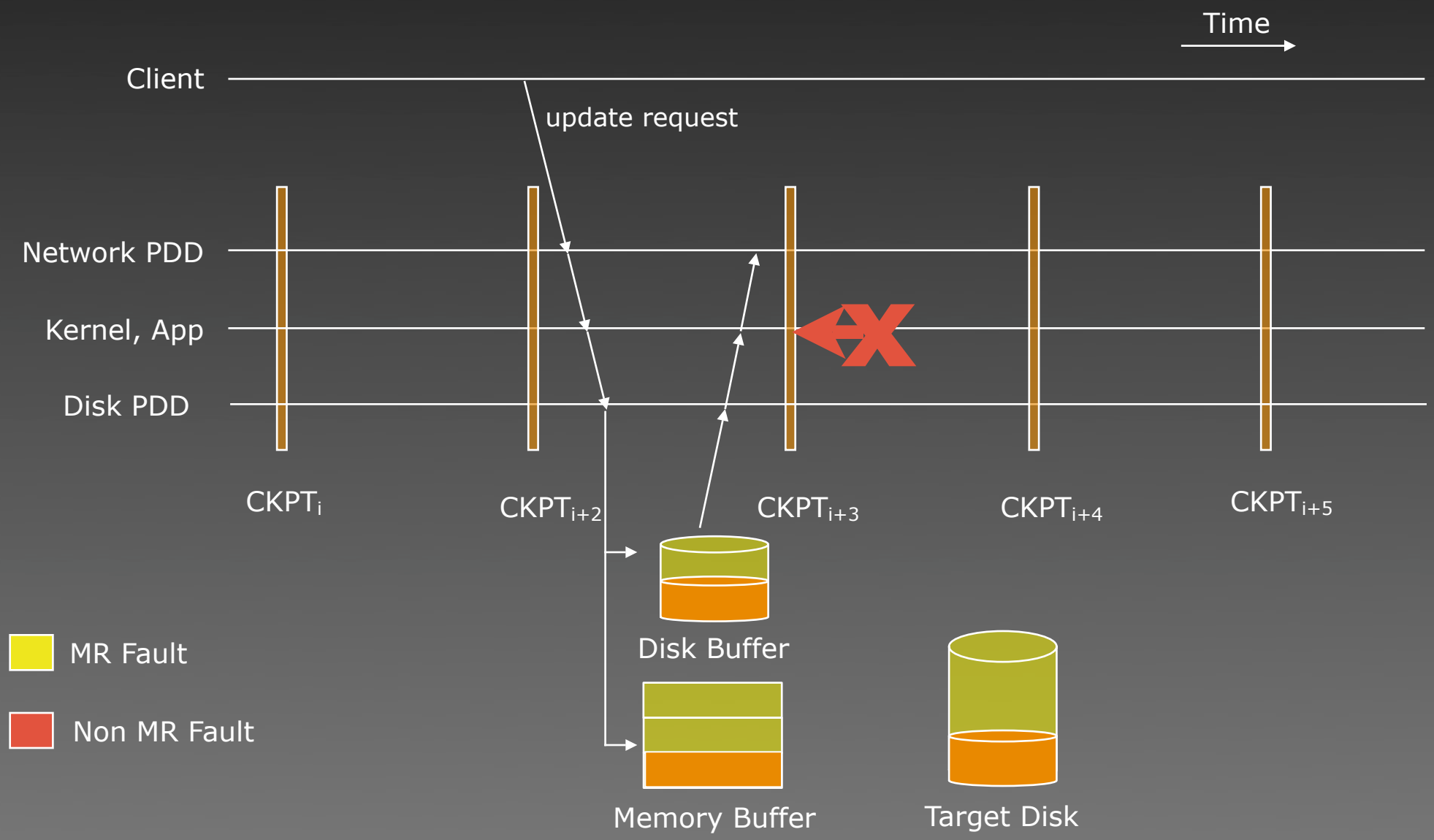
Disk PDD


Time
Client
update request
response
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
MR Fault
Memory Buffer
Target Disk

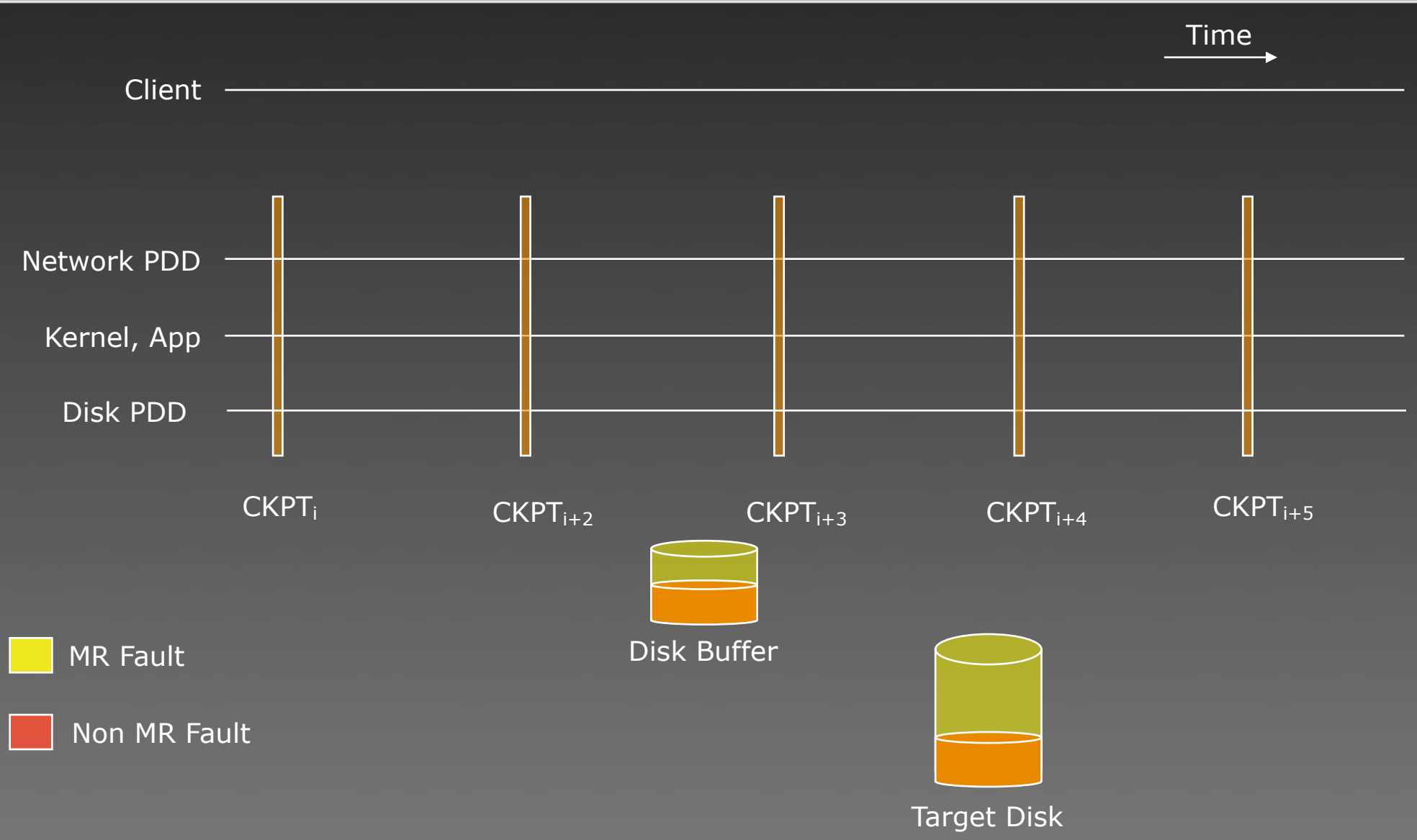
Disk PDD


Time
Client
update request
response
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk
MR Fault
Non MR Fault

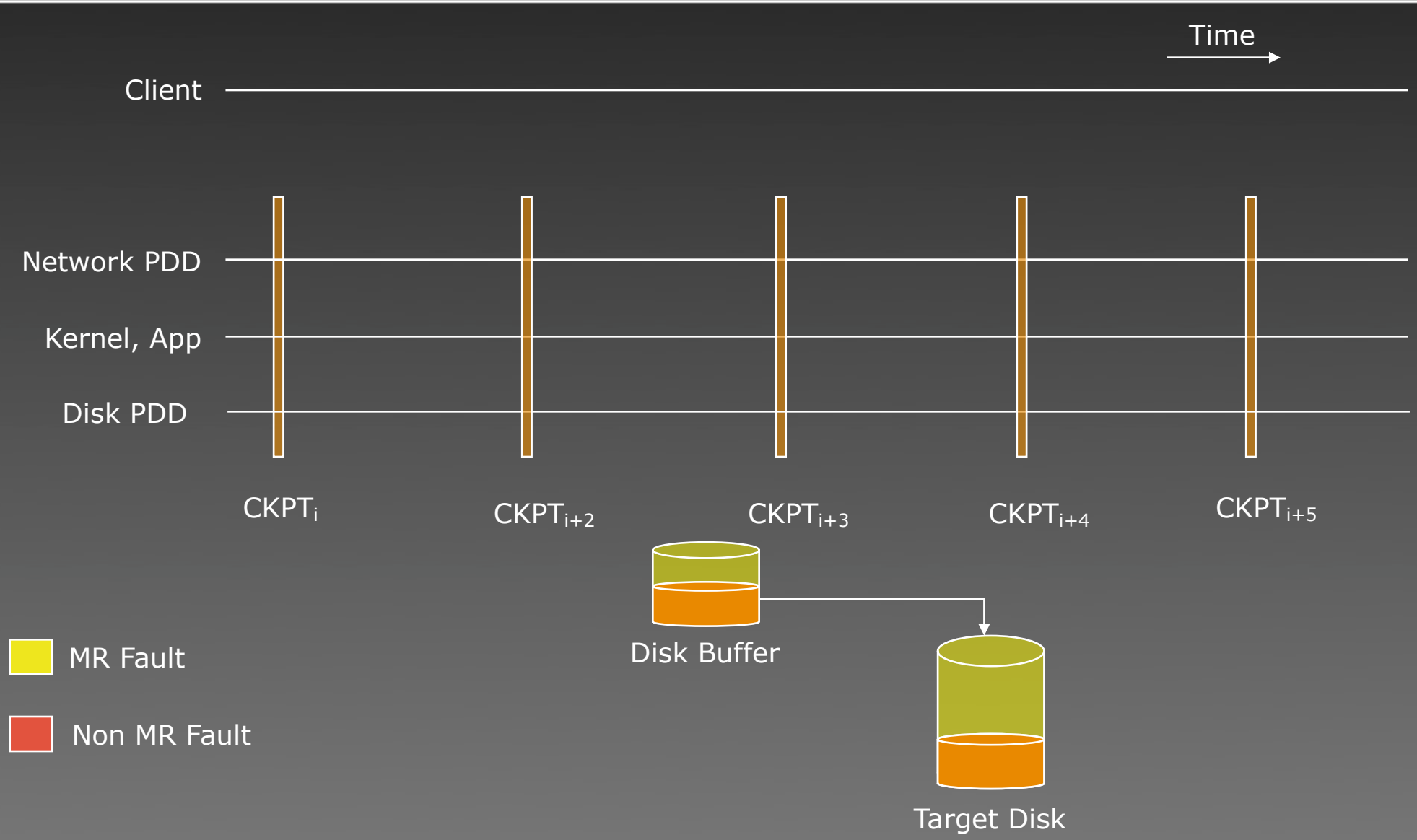
Disk PDD


Time
Client
update request
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Memory Buffer
Target Disk
MR Fault
Non MR Fault

Disk PDD

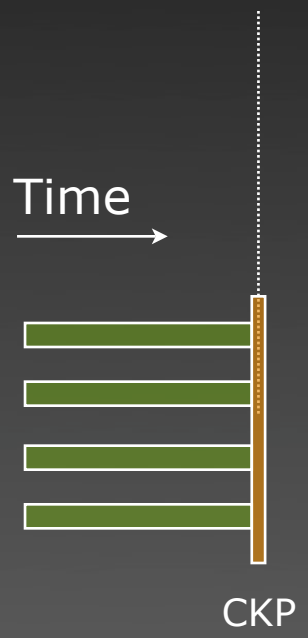

Time
Client
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Target Disk
MR Fault
Non MR Fault

Disk PDD


Time
Client
Network PDD
Kernel, App
Disk PDD
CKPTi
CKPTi+2
CKPTi+3
CKPTi+4
CKPTi+5
Disk Buffer
Target Disk
MR Fault
Non MR Fault

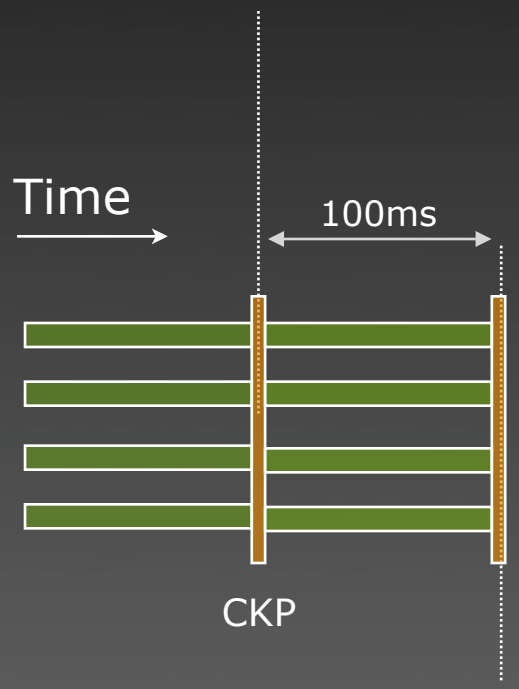
Recovery: ReVive + ReViveI/O

Recovery: ReVive + ReViveI/O



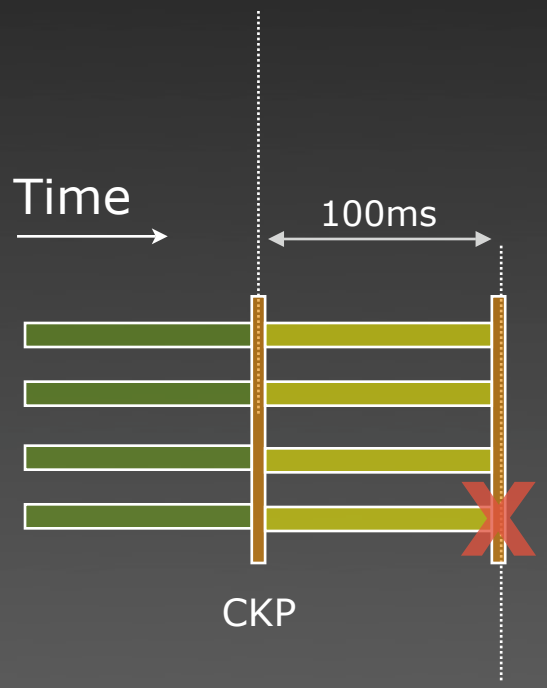
Useful Work

Recovery: ReVive + ReViveI/O



Useful Work

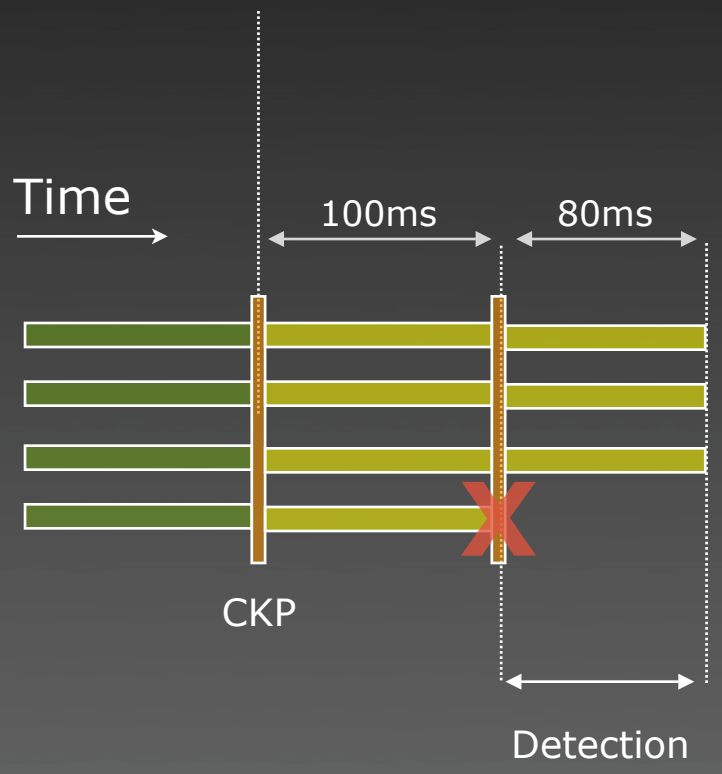
Recovery: ReVive + ReViveI/O



Useful Work

Lost Work

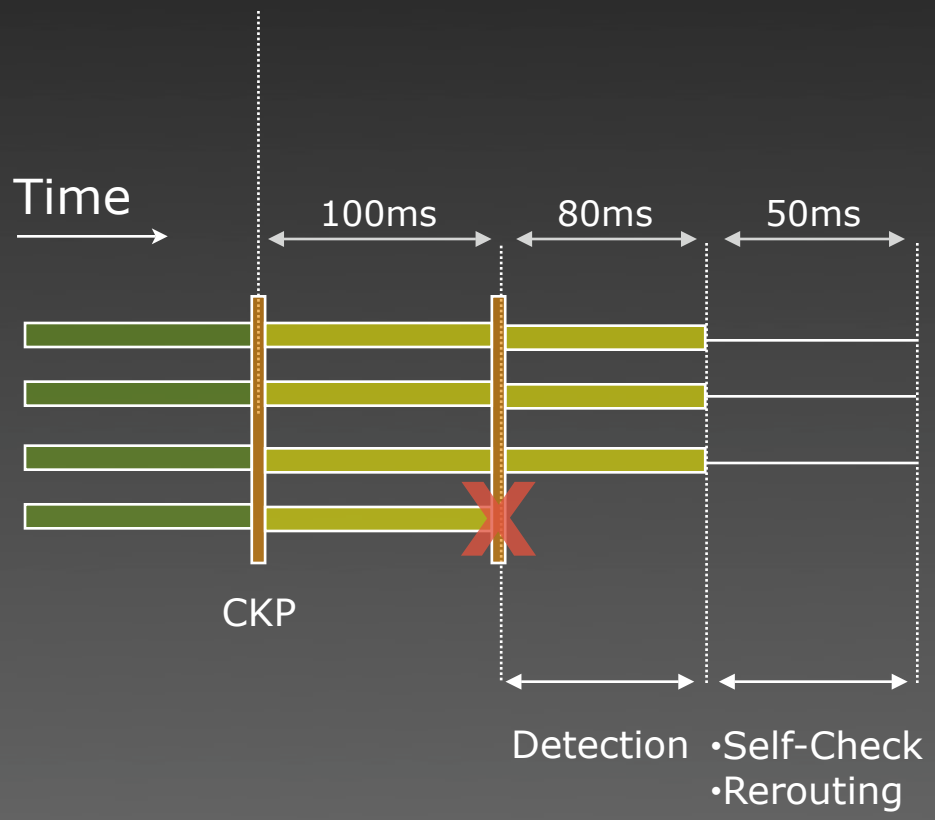
Recovery: ReVive + ReViveI/O



Useful Work

Lost Work

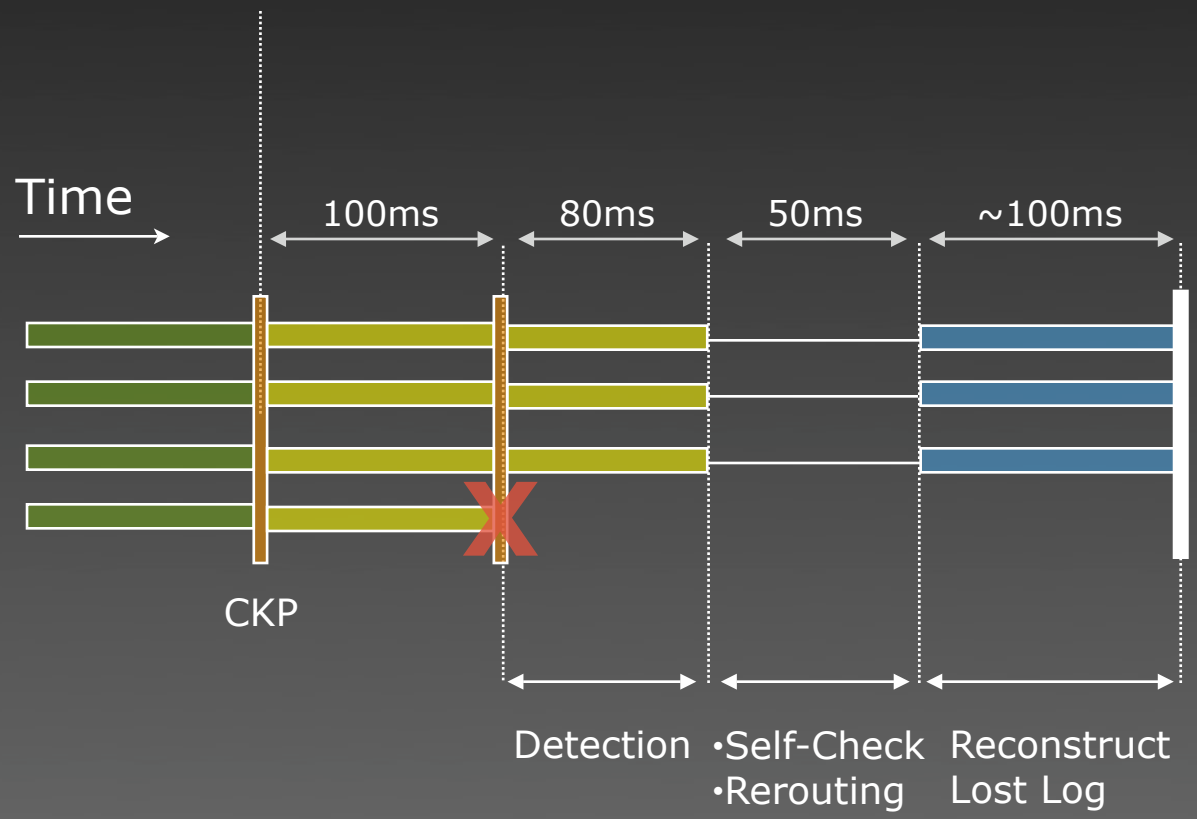
Recovery: ReVive + ReViveI/O



Useful Work

Lost Work

Recovery: ReVive + ReViveI/O

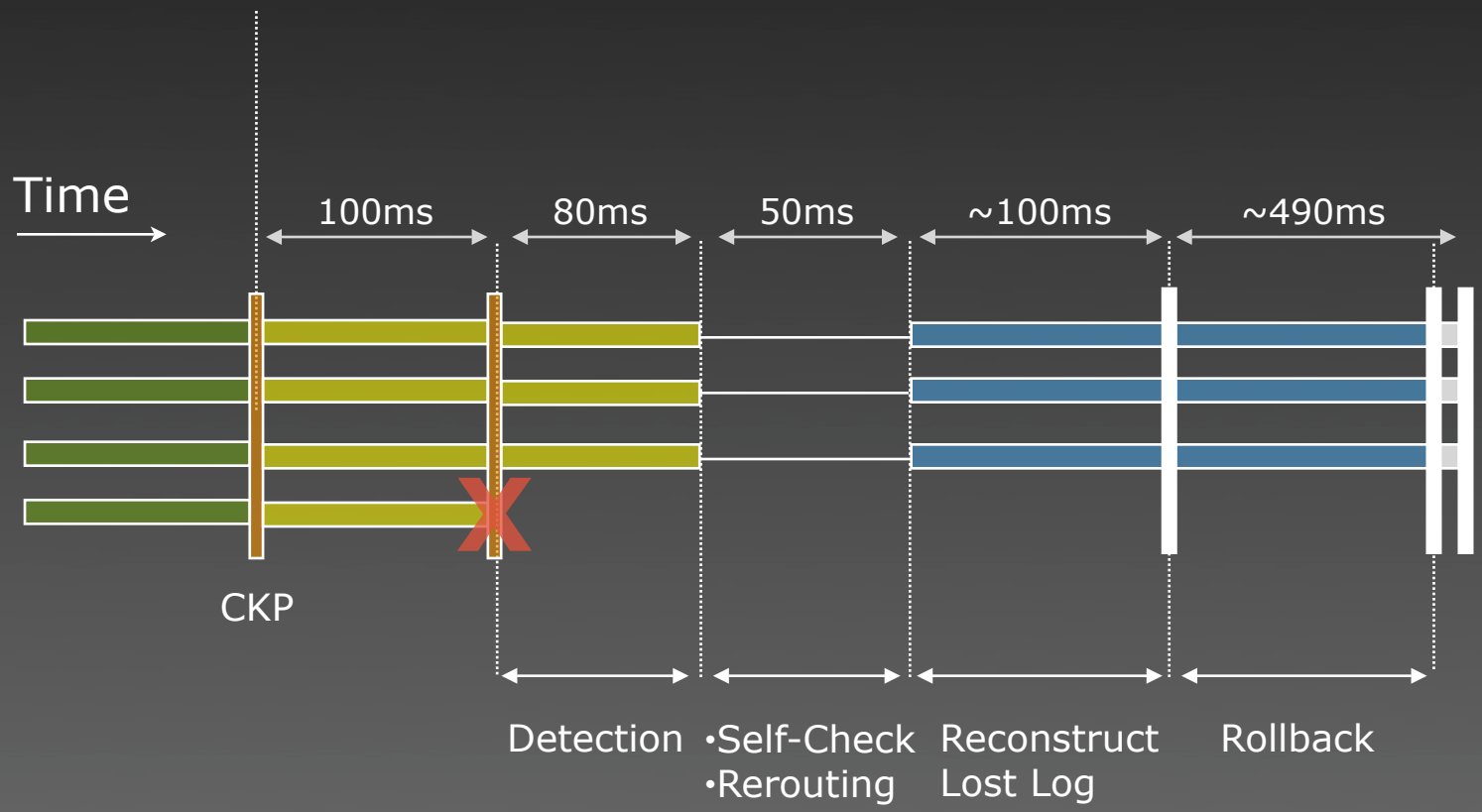


Useful Work

Lost Work

ReVive Recovery

Recovery: ReVive + ReViveI/O

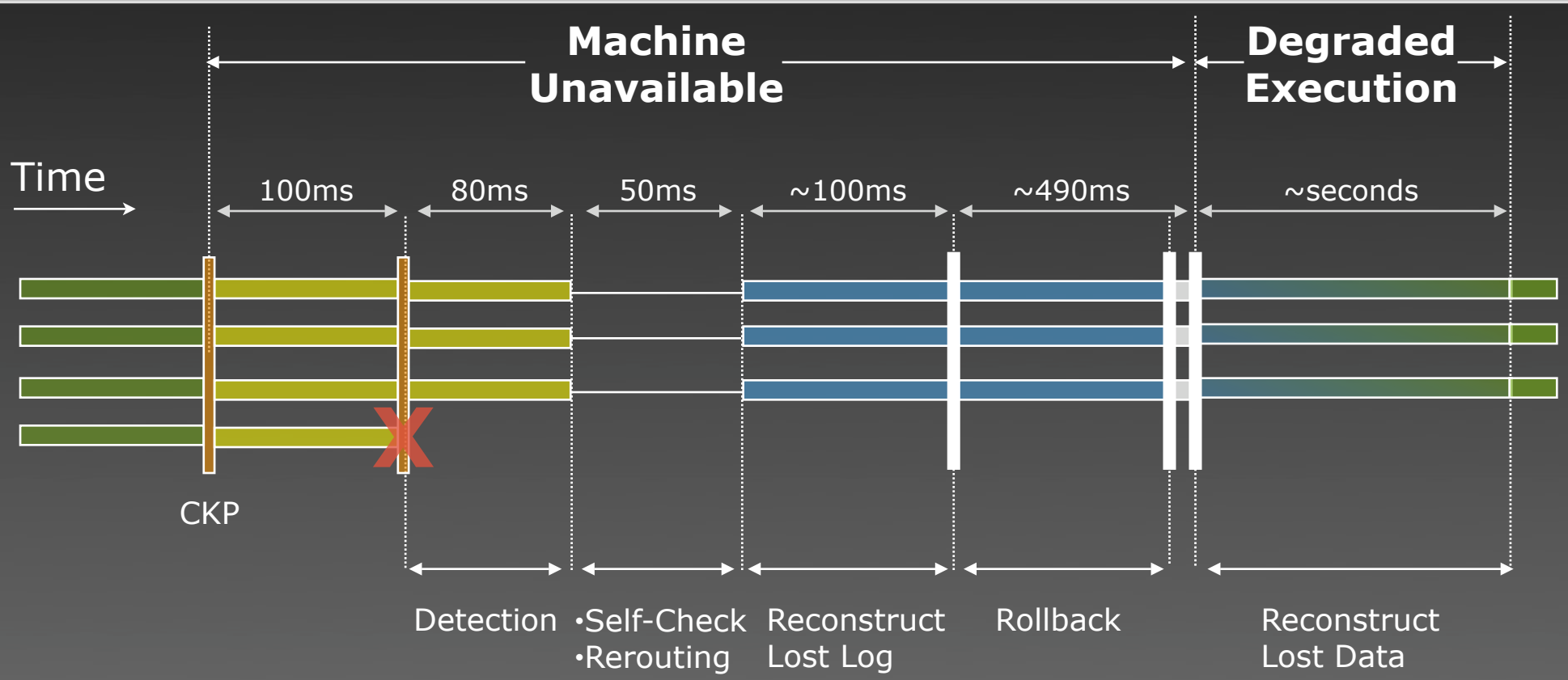


Useful Work

Lost Work

ReVive Recovery

Recovery: ReVive + ReViveI/O

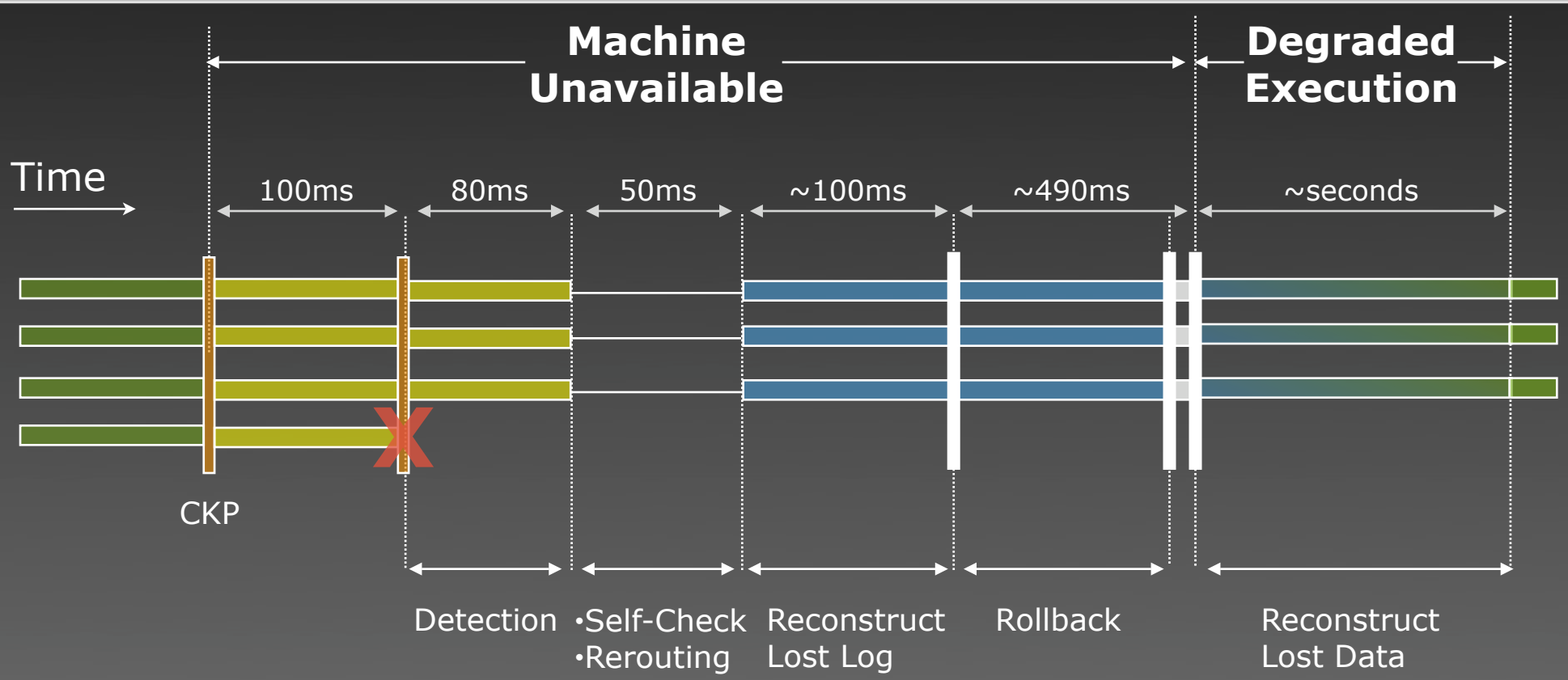


Useful Work

Lost Work

ReVive Recovery

Recovery: ReVive + ReViveI/O



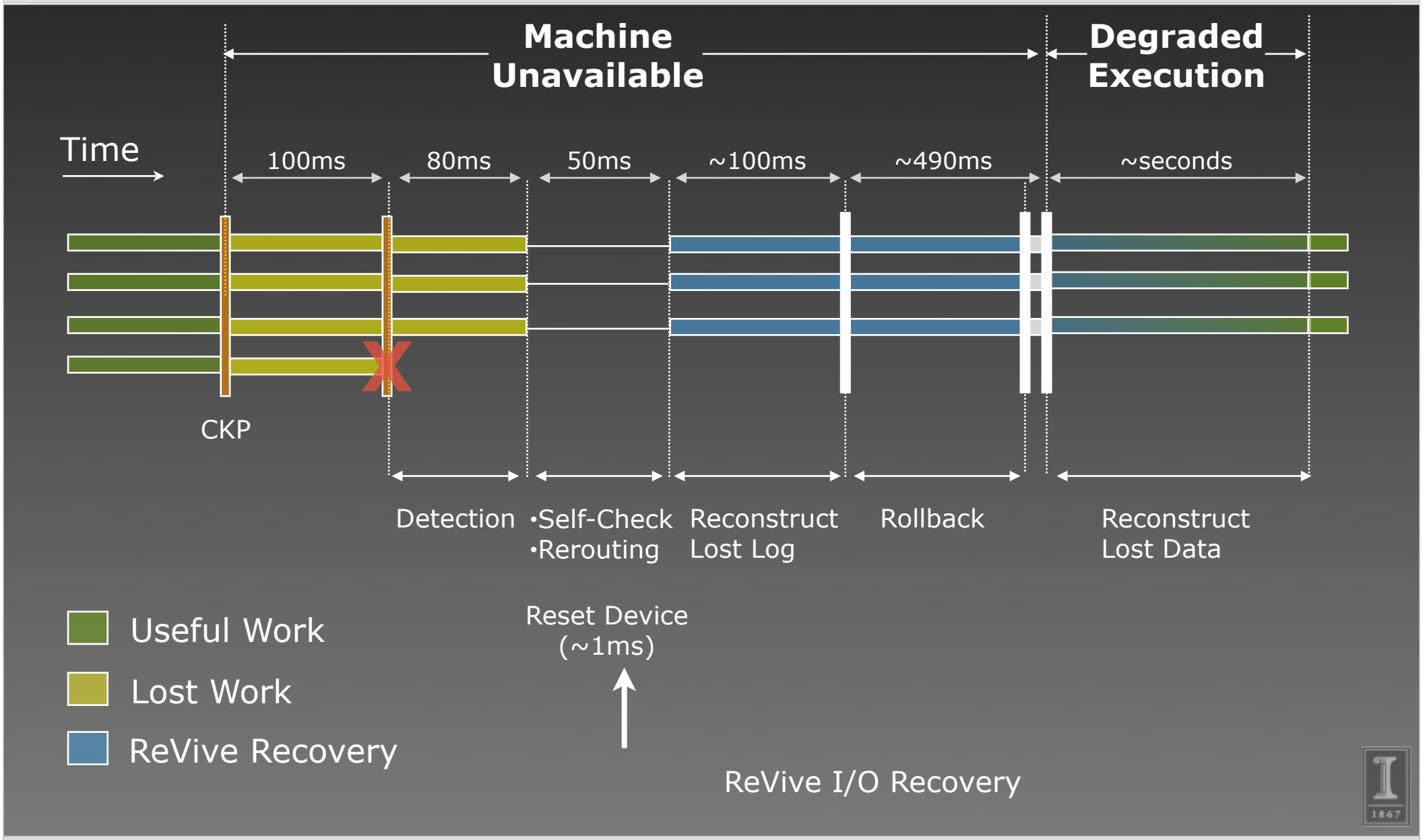
Useful Work

Lost Work

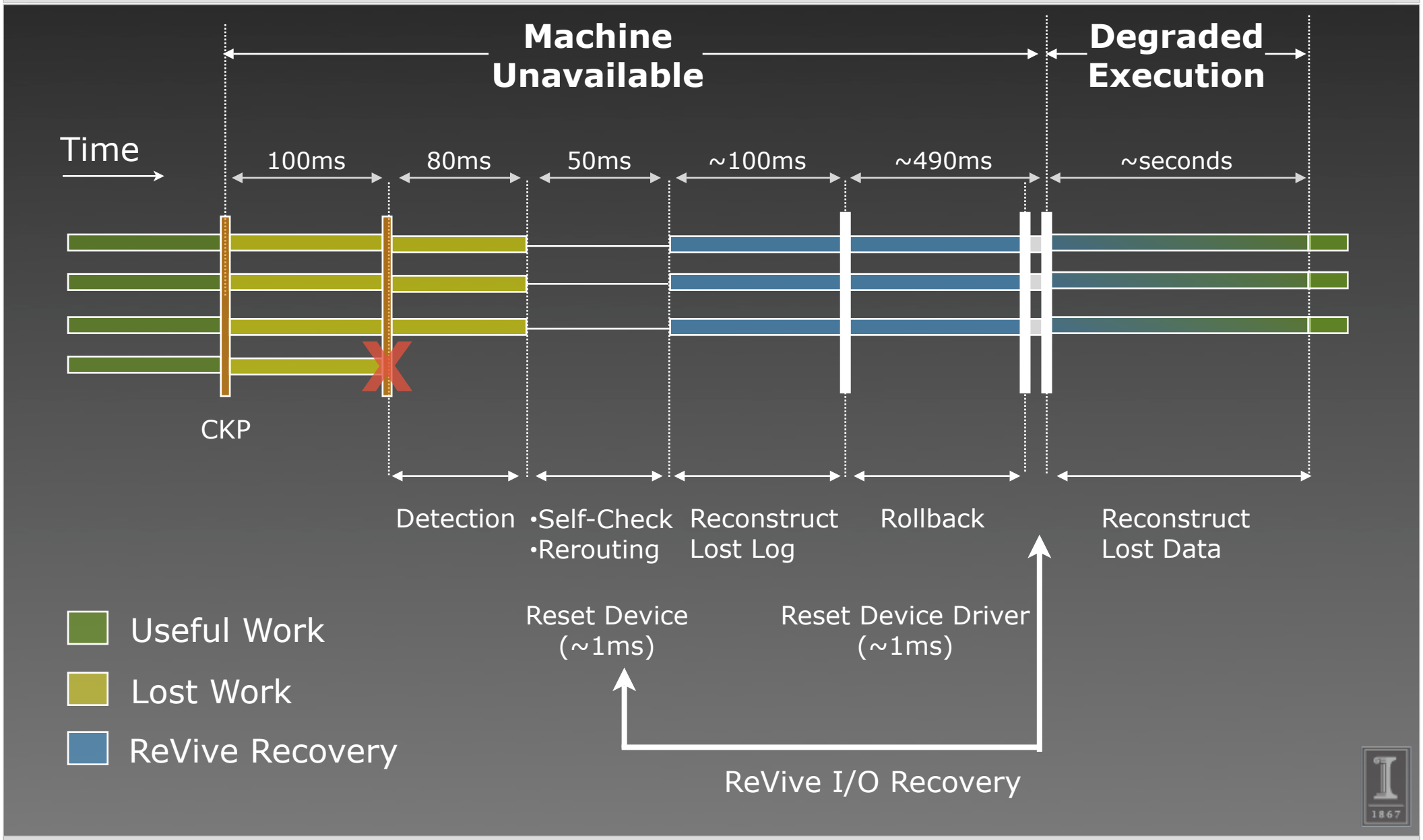
ReVive Recovery

ReVive I/O Recovery

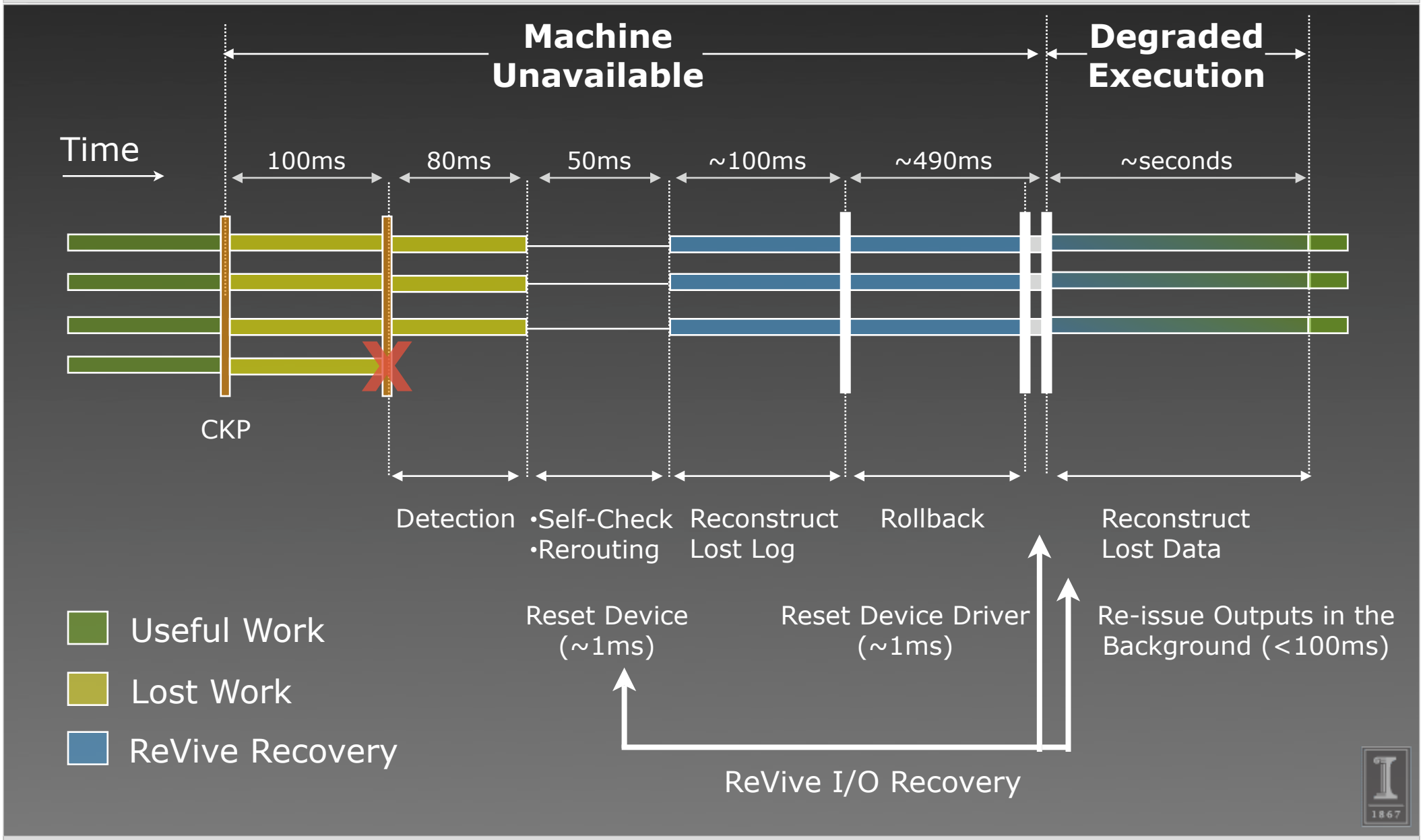
Recovery: ReVive + ReViveI/O



Recovery: ReVive + ReViveI/O



Recovery: ReVive + ReViveI/O



Experiments

Experiments

Experiments

Clients

- Dual 1.5 GHz Athlon
- Gigabit Ethernet
- 1 GB RAM

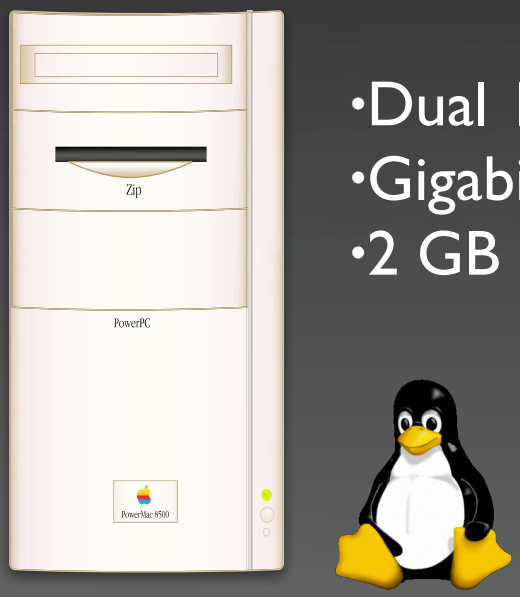
Experiments

Clients

- Dual 1.5 GHz Athlon
- Gigabit Ethernet
- 1 GB RAM

Server

- Dual 1.5 GHz Athlon
- Gigabit Ethernet
- 2 GB RAM



+ **ReVive** PDD

Throughput-oriented workload

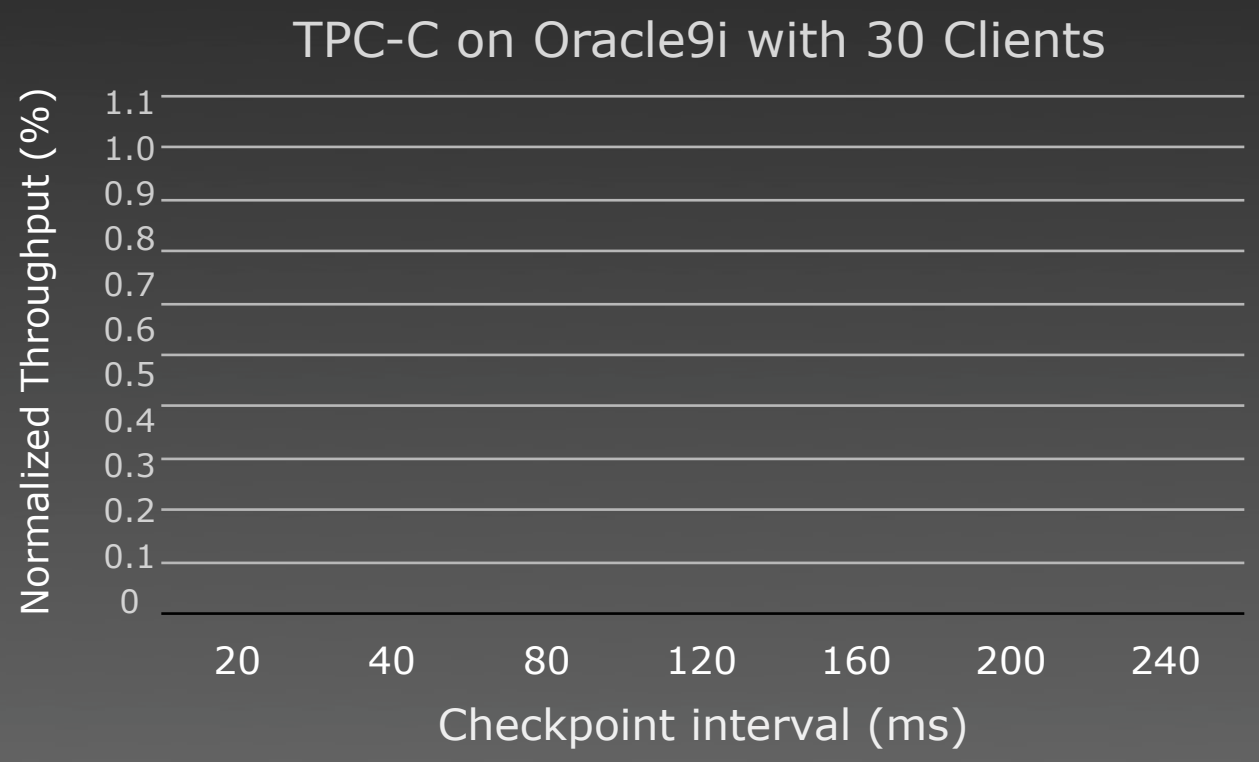
TPC-C on Oracle9i with 30 Clients

Normalized Throughput (%)

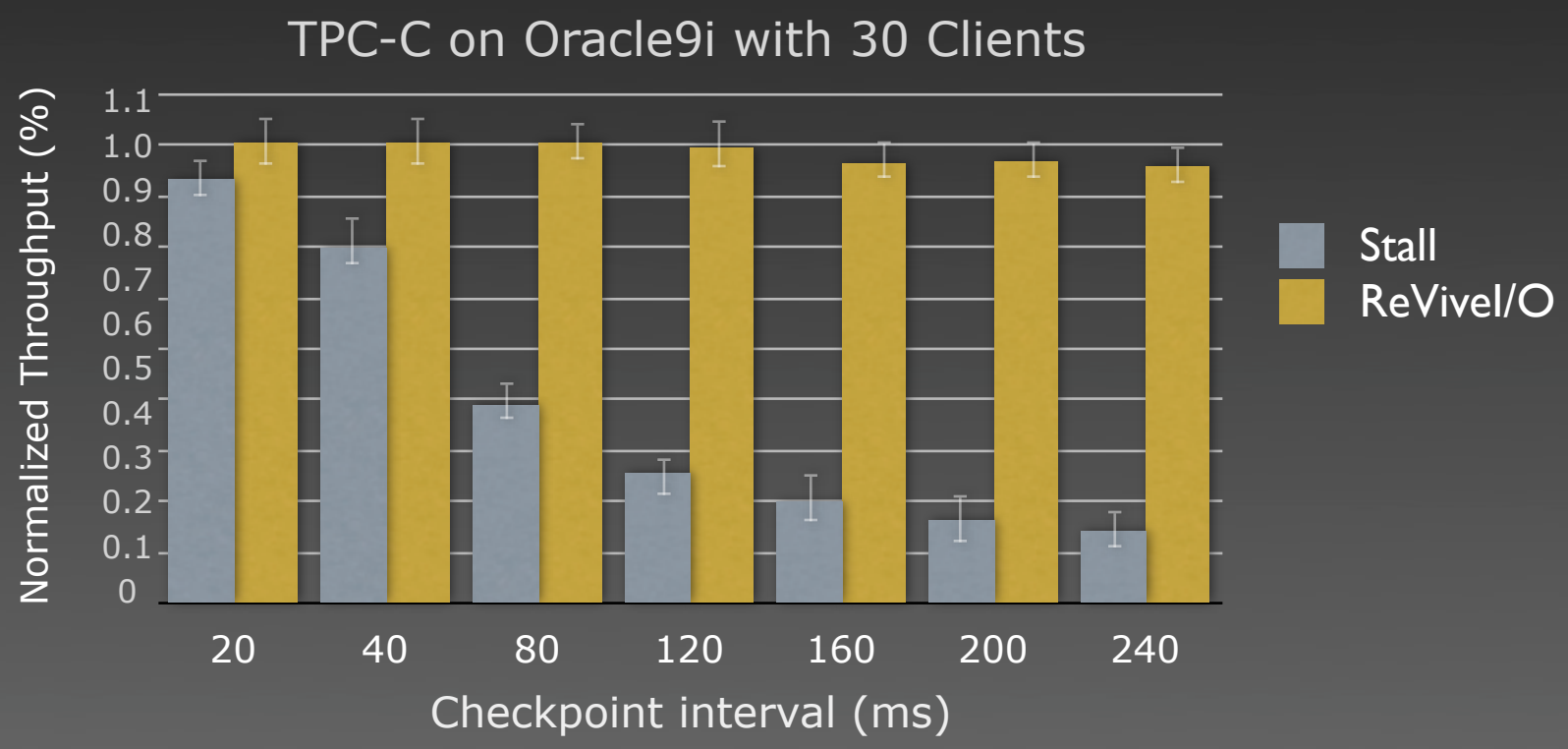
1.1
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0

Checkpoint interval (ms)

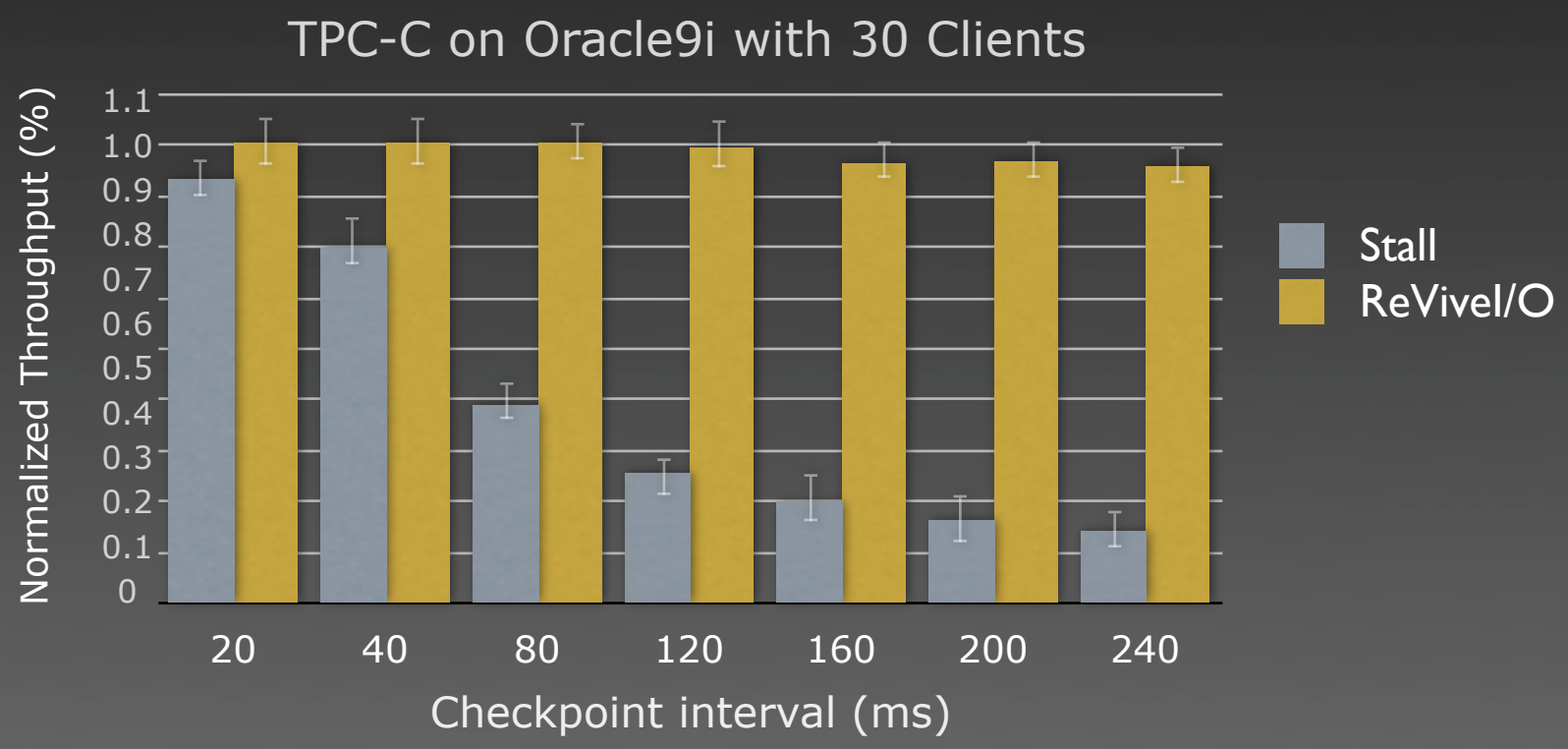
Throughput-oriented workload



Throughput-oriented workload

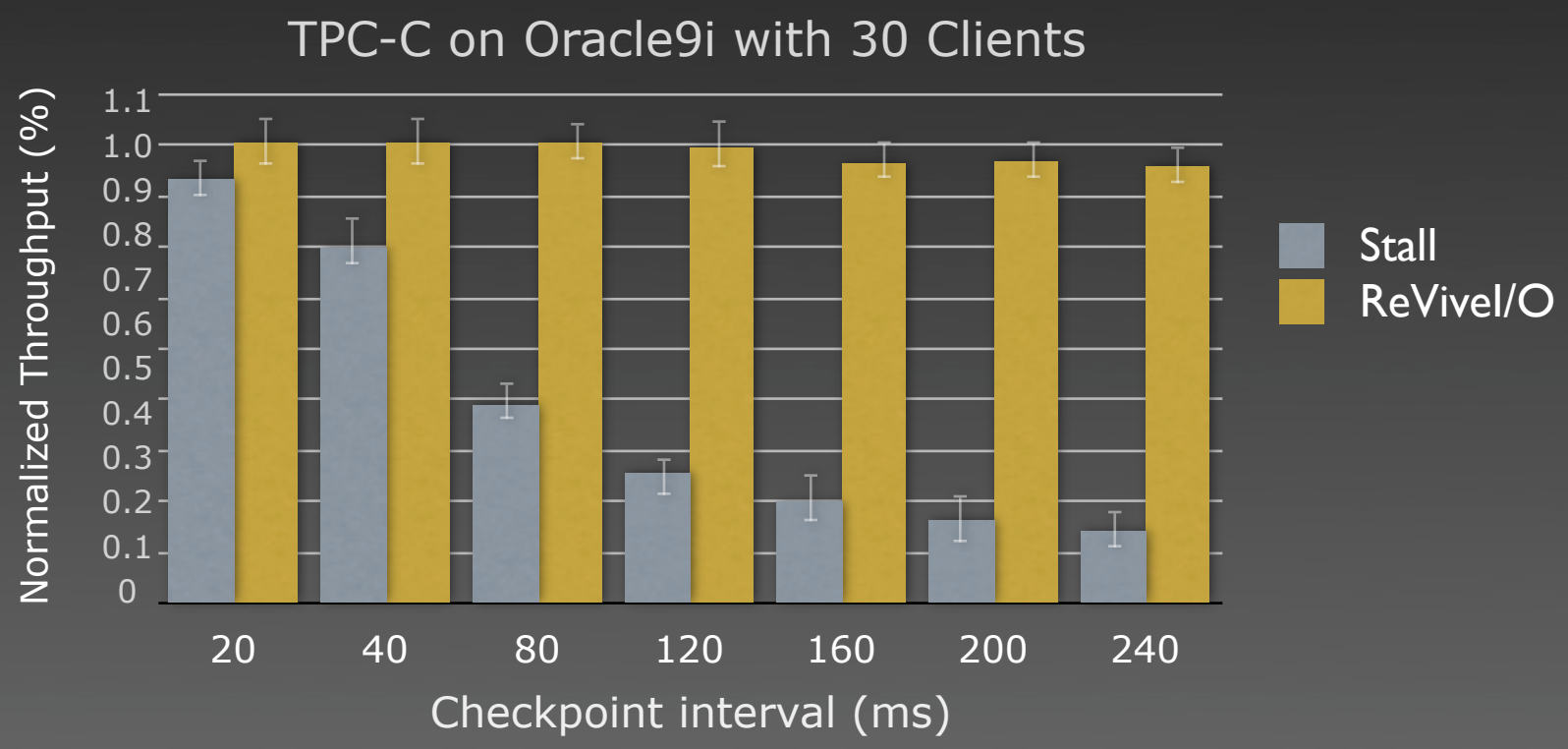


Throughput-oriented workload



- Stalling scheme (simply delaying writes) incurs too much overhead

Throughput-oriented workload



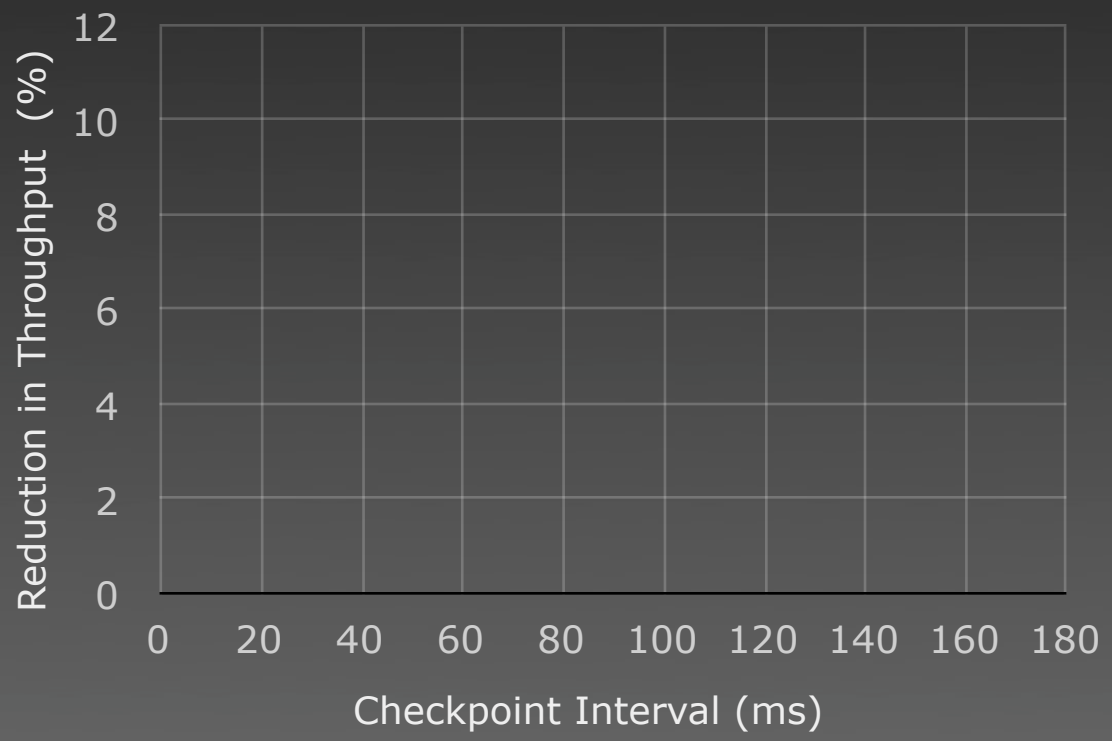
- Stalling scheme (simply delaying writes) incurs too much overhead
- ReViveI/O's overhead is only 0.1% for T<120 ms

Integration with ReVive (for TPC-C)

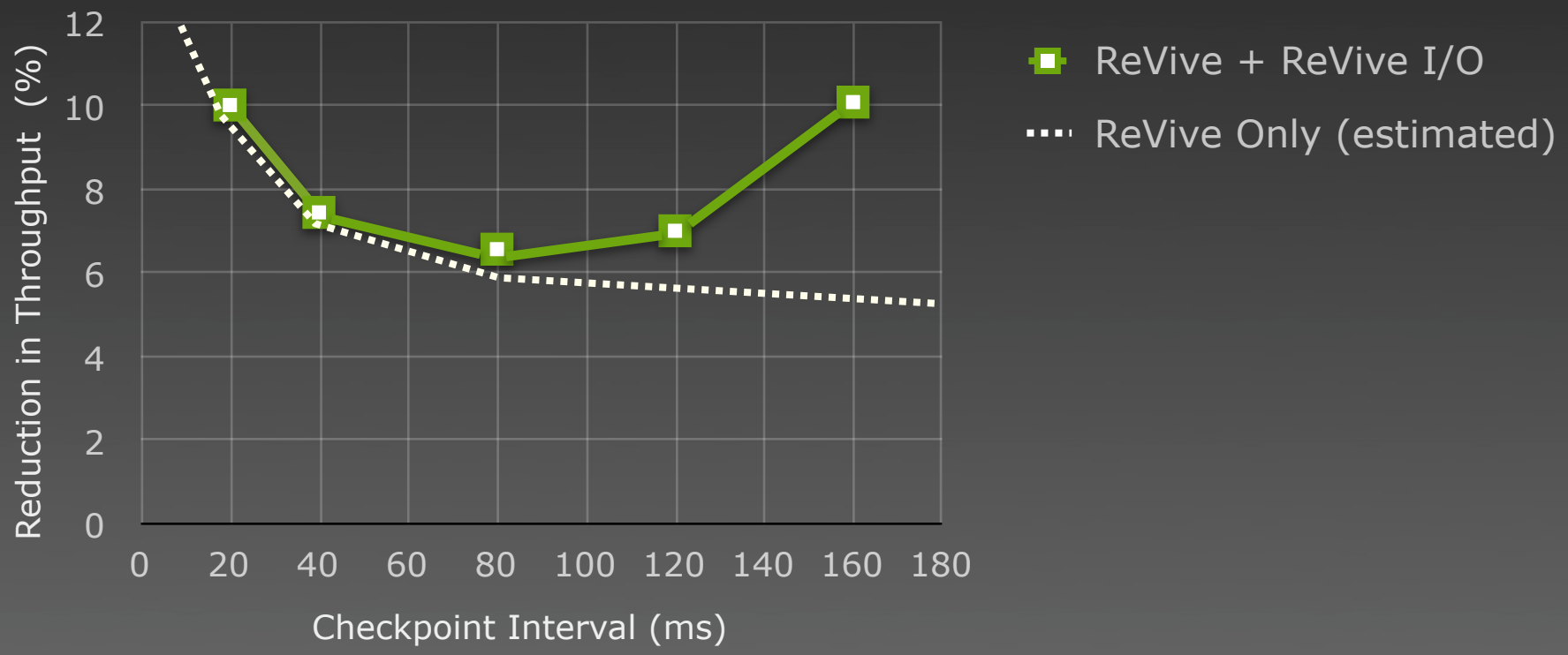
Reduction in Throughput (%)

Checkpoint Interval (ms)

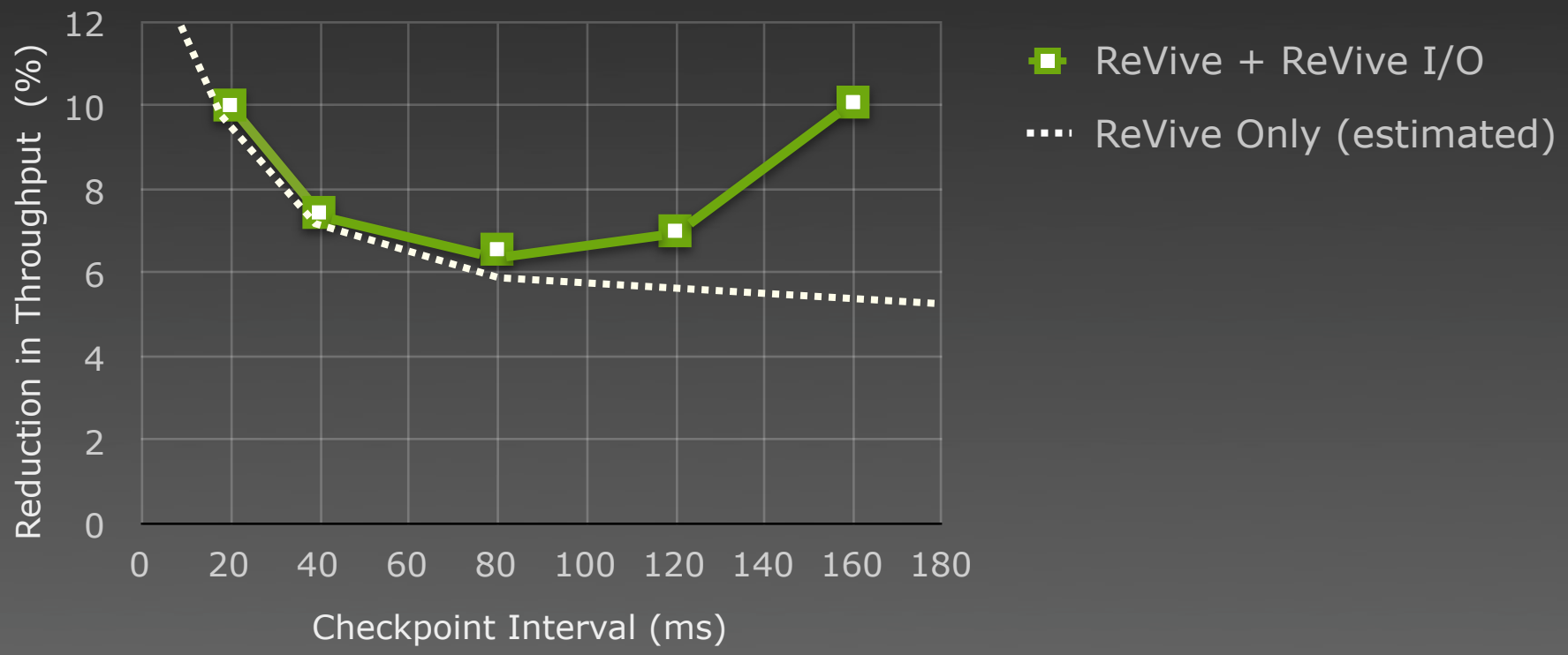
Integration with ReVive (for TPC-C)



Integration with ReVive (for TPC-C)

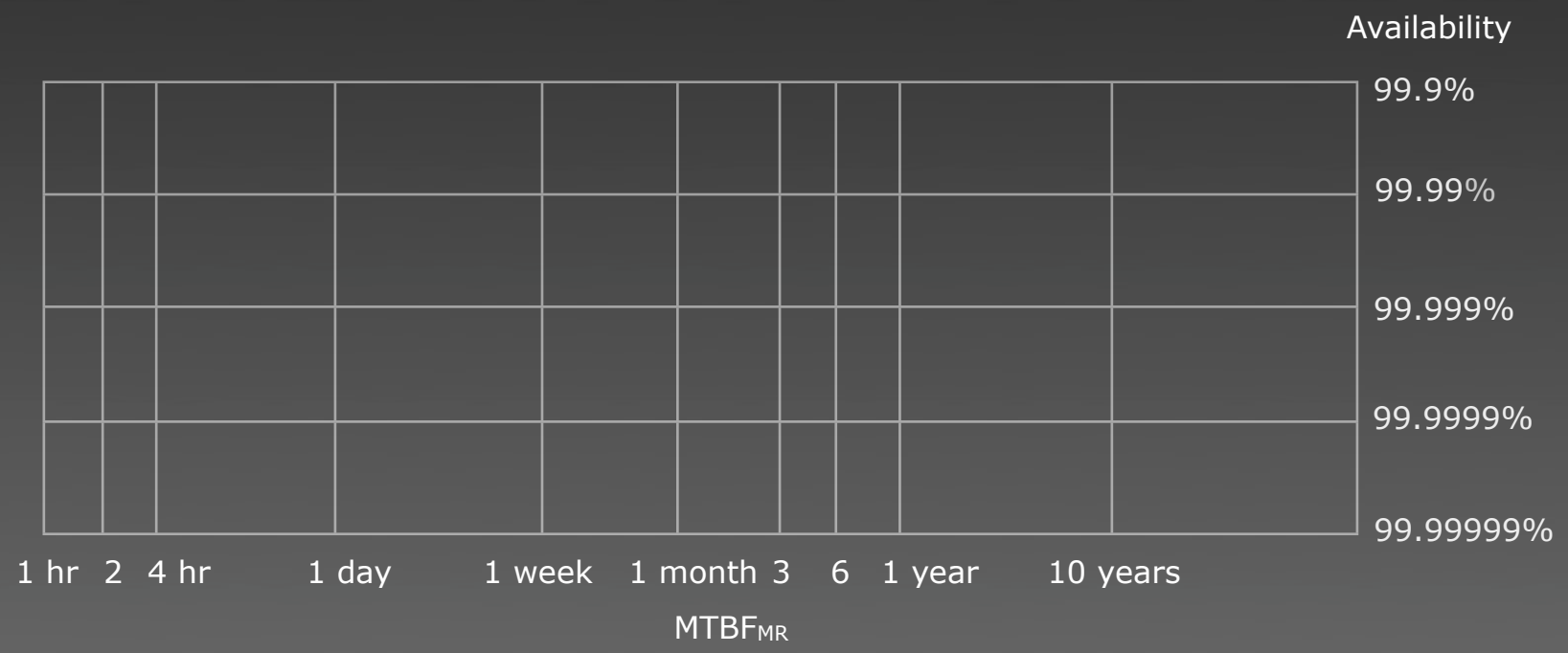


Integration with ReVive (for TPC-C)

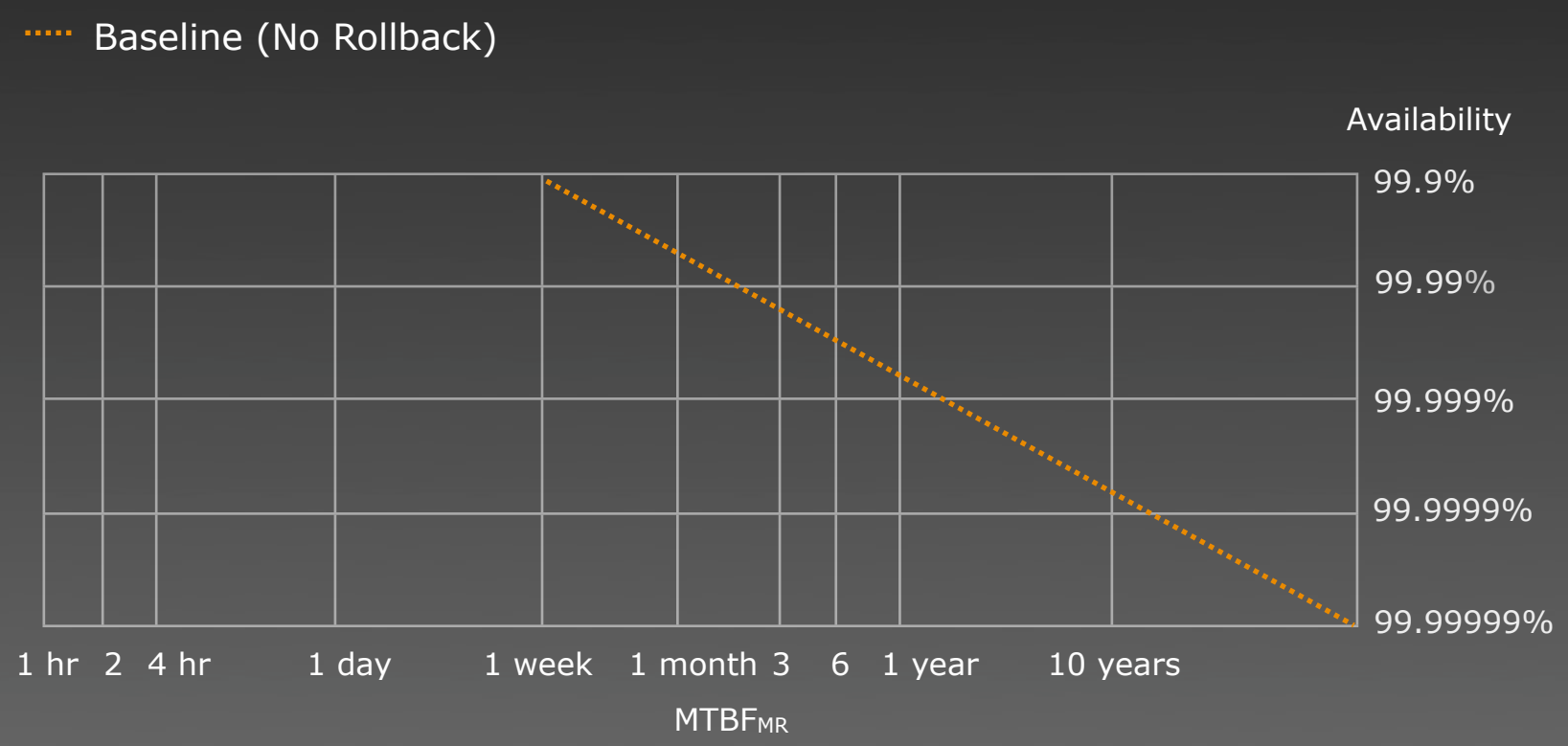


- ReVive + ReViveI/O: only 7% throughput reduction for 60 - 120 ms checkpoint intervals

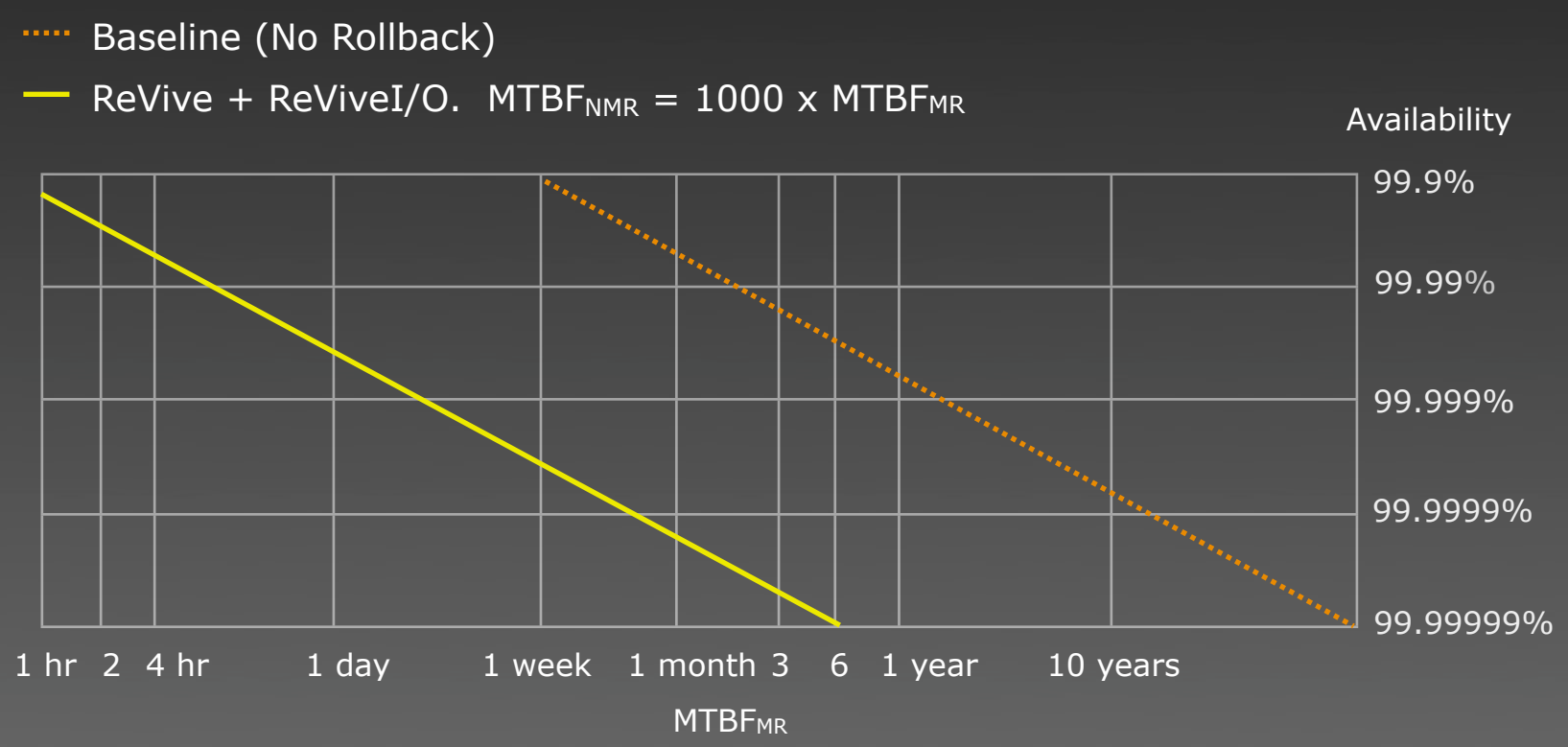
Availability



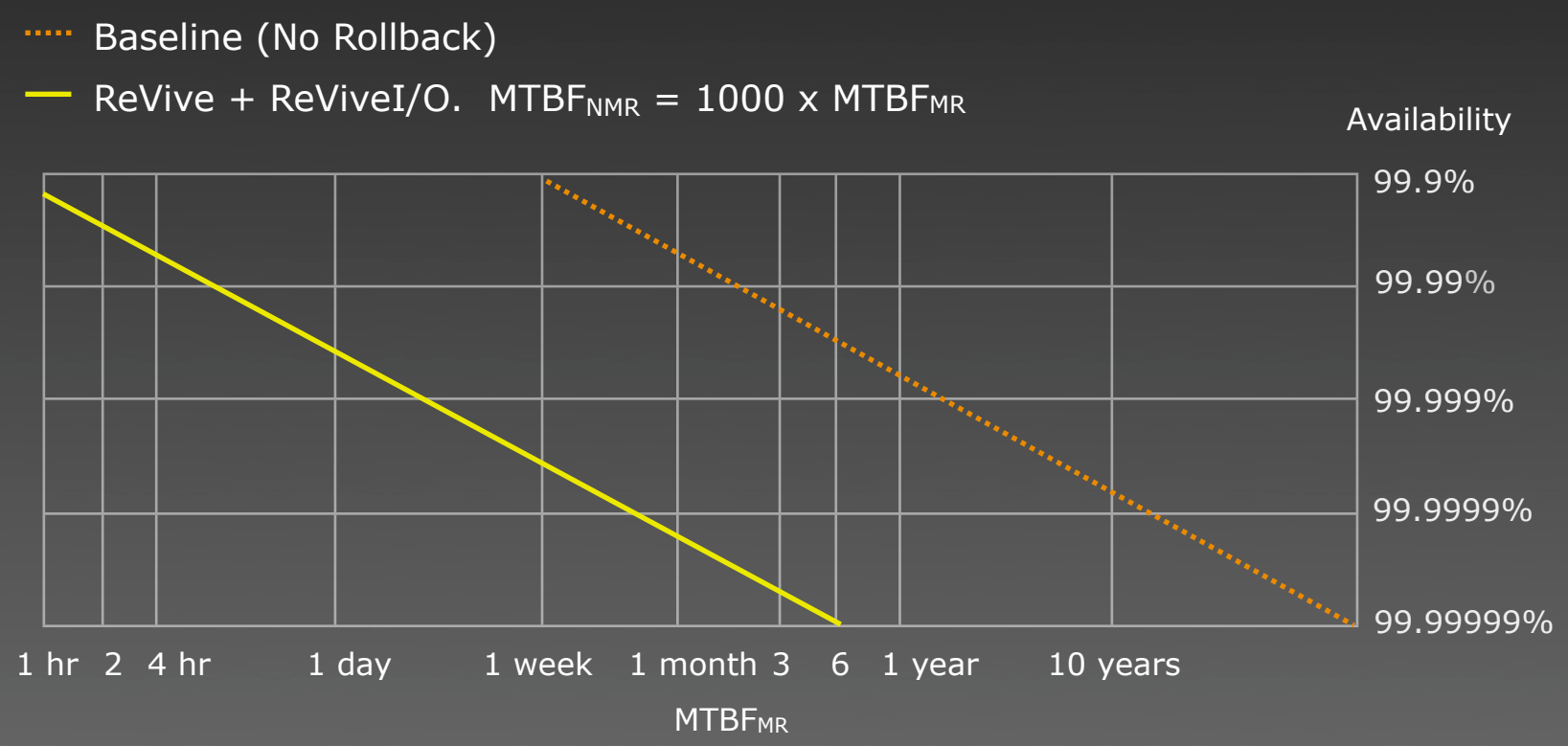
Availability



Availability



Availability



- ReVive I/O beats baseline thanks to its low recovery latency for MR faults

Conclusions

Conclusions

- Support for I/O undo/redo in memory-checkpointing systems:
  - Transparent to OS and applications
  - Fast recovery and negligible overhead during fault-free operation
  - Low space overhead (see paper)

Conclusions

- Support for I/O undo/redo in memory-checkpointing systems:
  - Transparent to OS and applications
  - Fast recovery and negligible overhead during fault-free operation
  - Low space overhead (see paper)
- Troughtput-oriented workloads:
  - < 1% overhead over ReVive

Conclusions

- Support for I/O undo/redo in memory-checkpointing systems:
  - Transparent to OS and applications
  - Fast recovery and negligible overhead during fault-free operation
  - Low space overhead (see paper)
- Troughtput-oriented workloads:
  - < 1% overhead over ReVive
- Latency-bound workloads:
  - Tolerable checkpoint interval is application dependent (see paper)

Conclusions

- Support for I/O undo/redo in memory-checkpointing systems:
  - Transparent to OS and applications
  - Fast recovery and negligible overhead during fault-free operation
  - Low space overhead (see paper)
- Troughtput-oriented workloads:
  - < 1% overhead over ReVive
- Latency-bound workloads:
  - Tolerable checkpoint interval is application dependent (see paper)
- ReViveI/O can be used for transactional memory

ReVive I/O:

Efficient Handling of I/O in Highly-Available Rollback-Recovery Servers

Jun Nakano[✢], Pablo Montesinos, Kourosh Gharachorloo[❄], Josep Torrellas

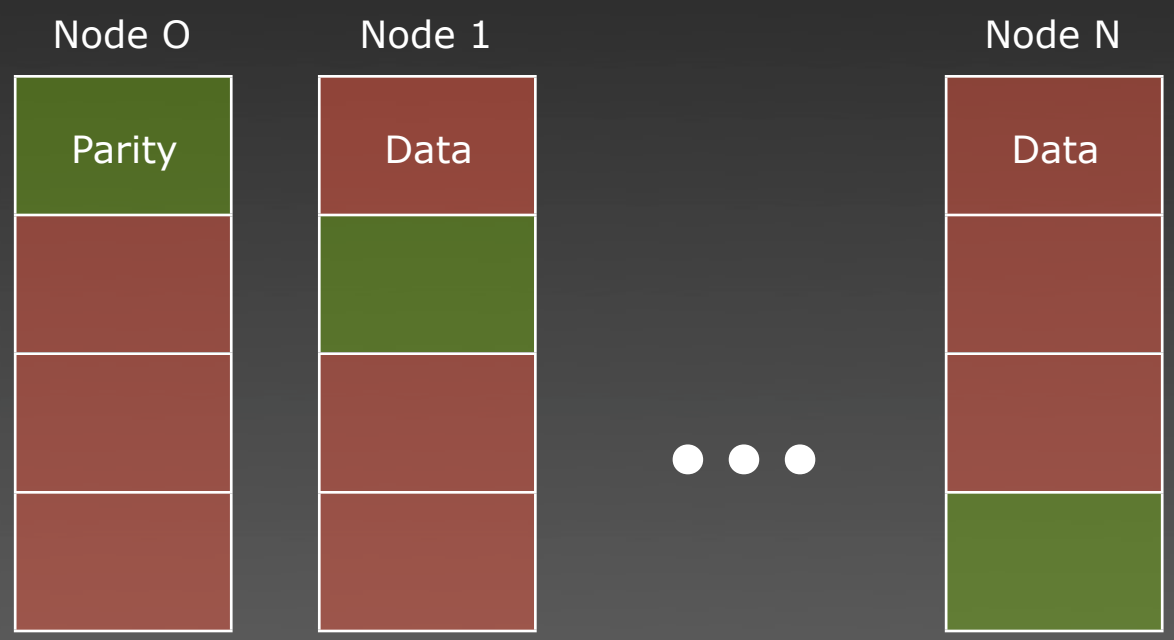
University of Illinois at Urbana-Champaign
[✢]IBM
[❄]Google

i-acoma group

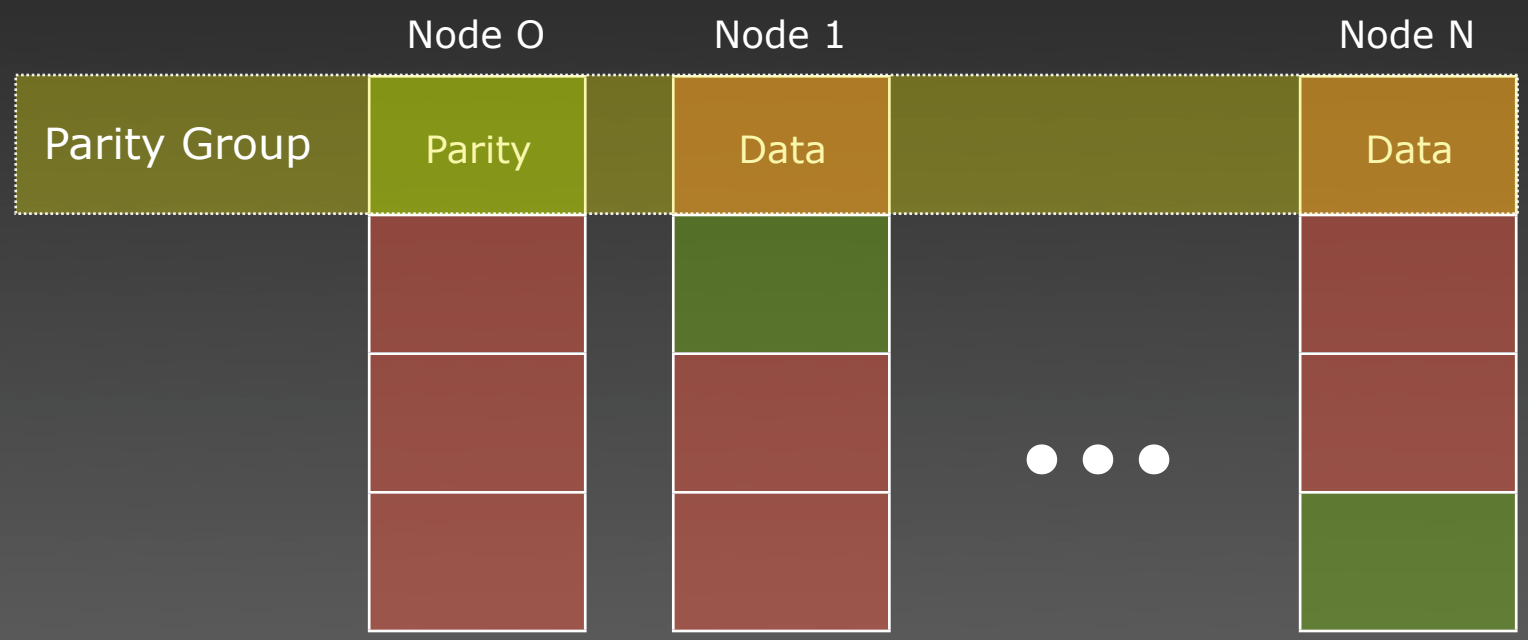
1867

Distributed N+1 Parity in ReVive

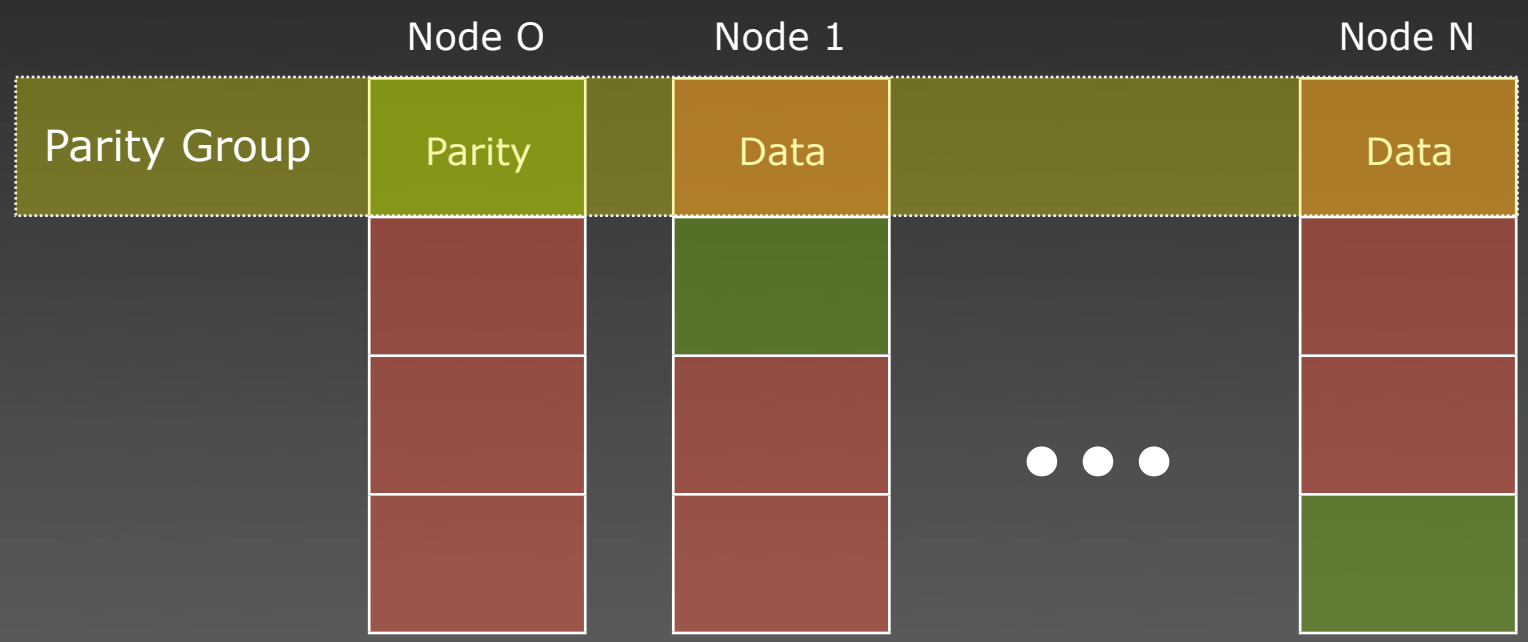
Distributed N+1 Parity in ReVive



Distributed N+1 Parity in ReVive



Distributed N+1 Parity in ReVive



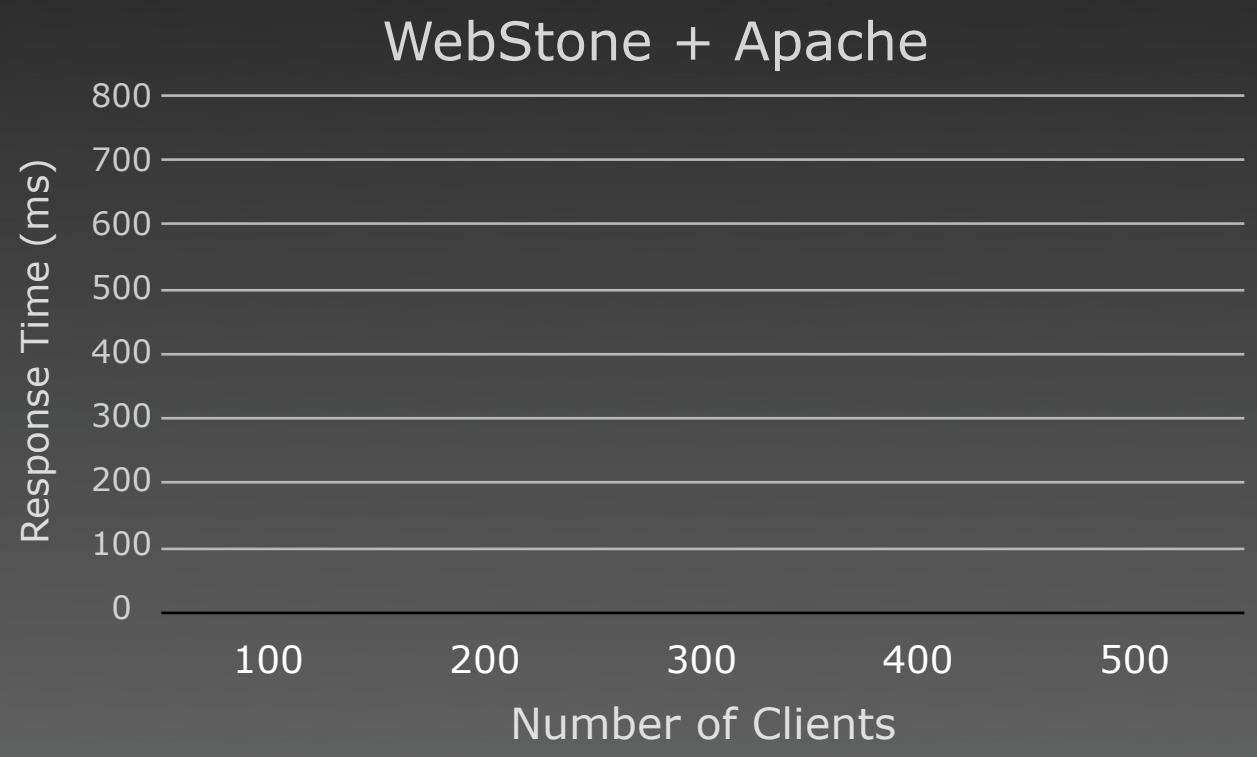
- Allocation granularity: page
- Update granularity: cache line
- Distributed parity: can recover from loss of one node

Space Overhead

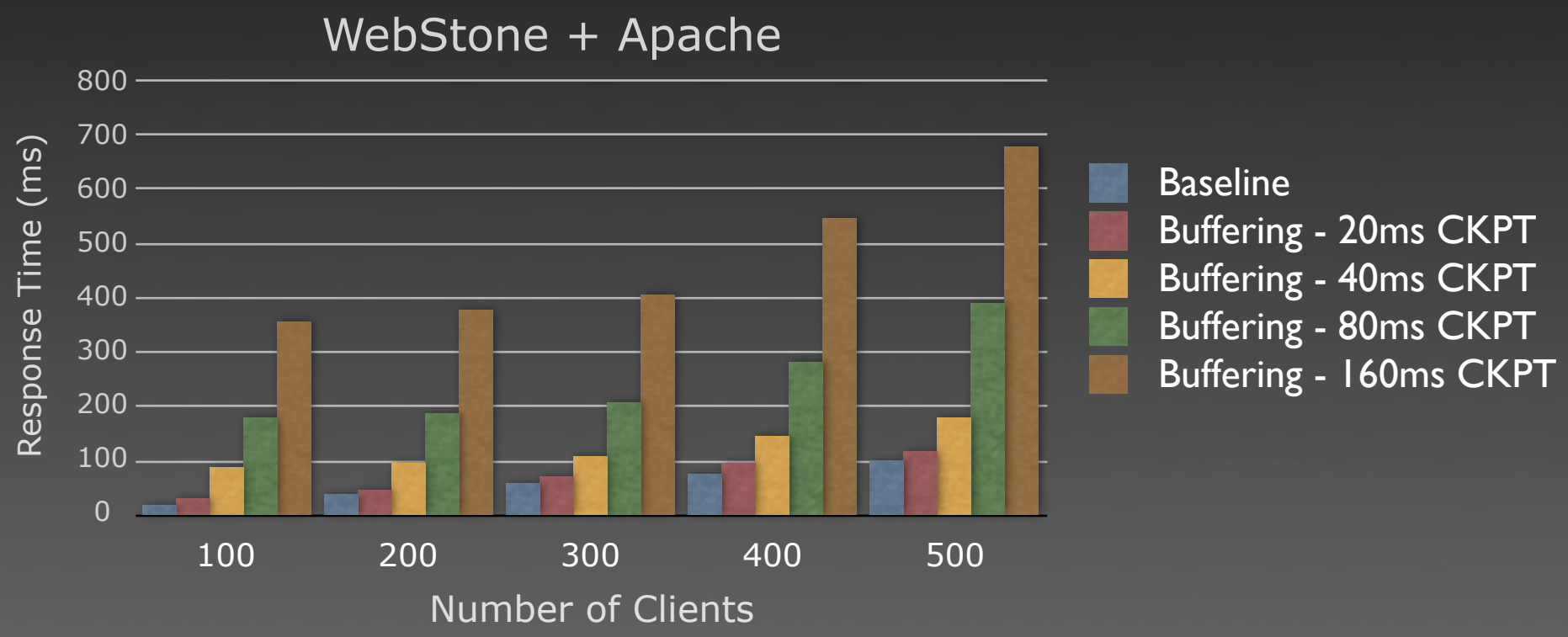
- Network PDD:
  - Need to increase OS's TCP sliding window size
  - e.g., Gigabit Ethernet, T = 100 ms → 12 MB
- Disk PDD (buffering):
  - Disk PDD needs private memory area for buffering
  - e.g., 40 MB/s disk, T = 100 ms → 4 MB

Latency-oriented workload

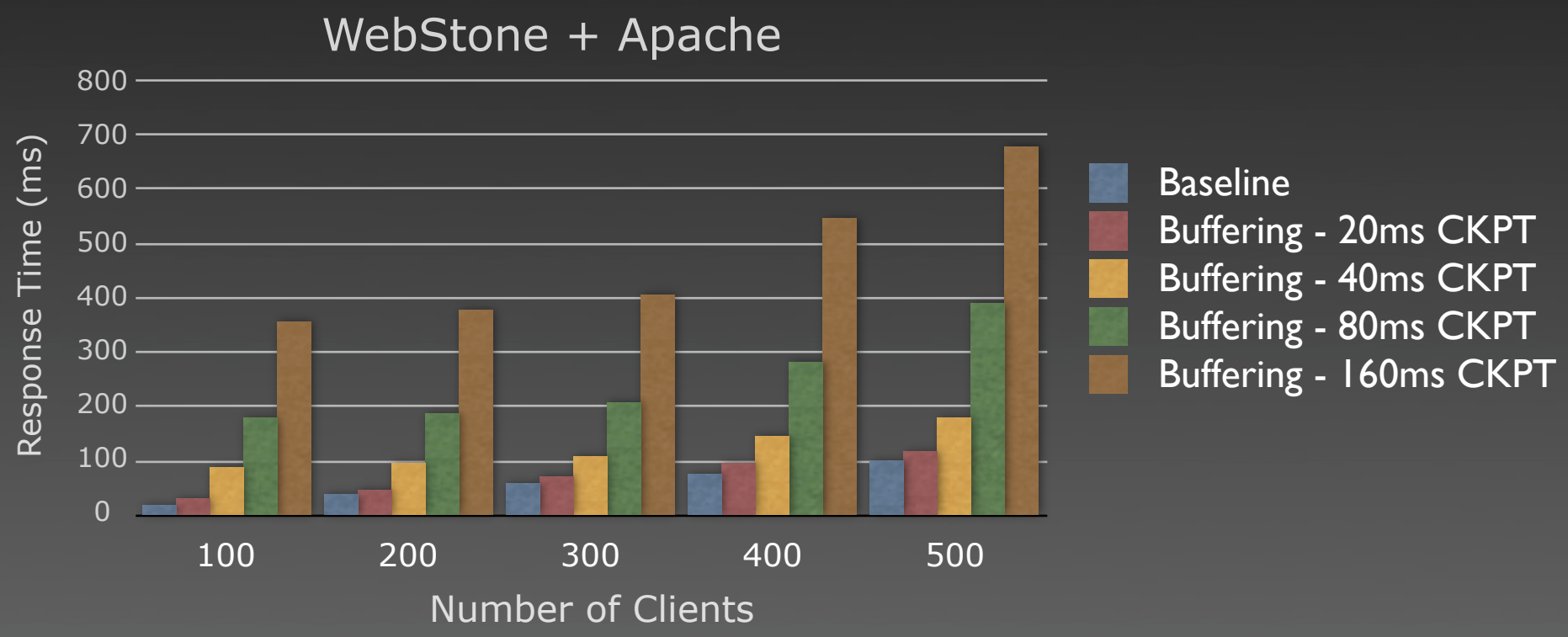
Latency-oriented workload



Latency-oriented workload



Latency-oriented workload



- Tolerable checkpoint interval is application dependent
- Response times up to 100ms are acceptable for interactive applications

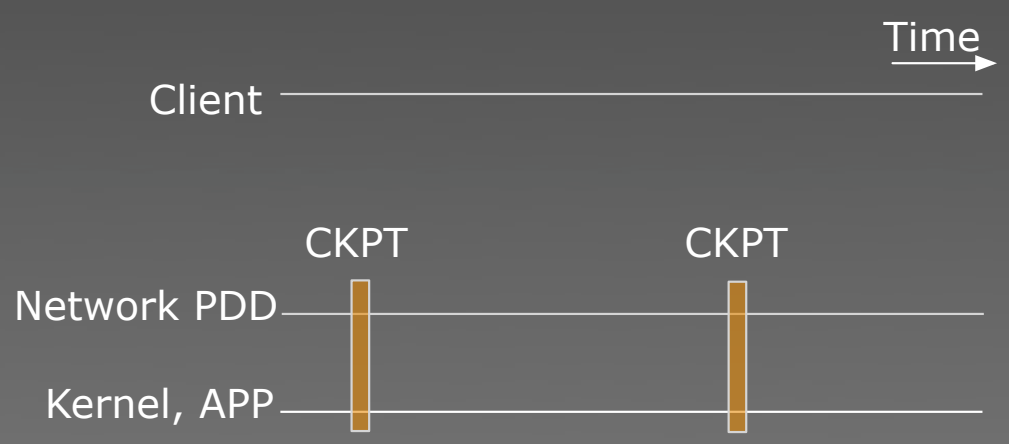
Disk PDD Schemes

Stall	On I/O write issue	Stalls the write
	After next CKPT	Writes data to disk
Logging	On I/O write issue	Copies old data elsewhere in disk Writes data in place
	After next CKPT	Deletes pointer to old data
Buffering	On I/O write issue	Writes data to disk buffer and mem
	After next CKPT	Copies data from mem to disk
Renaming	On I/O write issue	Writes data to renamed disk location Saves old logical → physical mapping
	After next CKPT	Delete old mapping

Network PDD and TCP

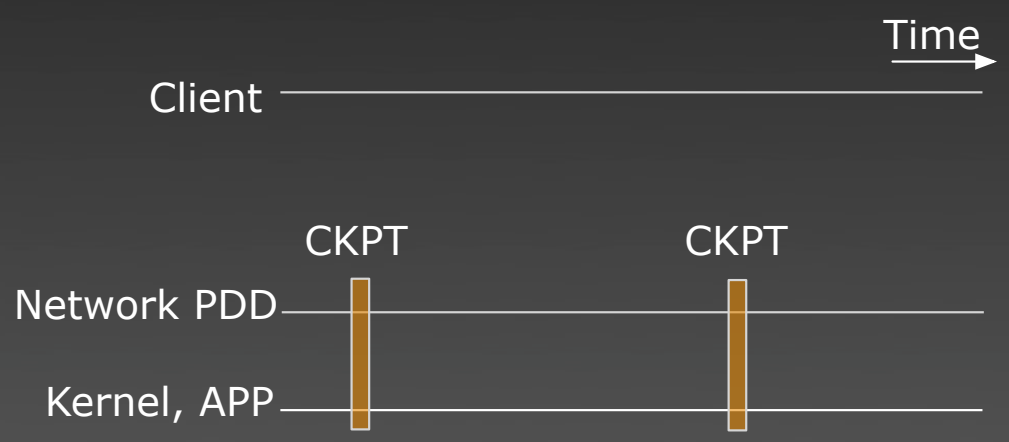
- Resend packets after rollback
- TCP eliminates duplicates

- Can avoid saving inputs replay
- Sender timeouts and resend

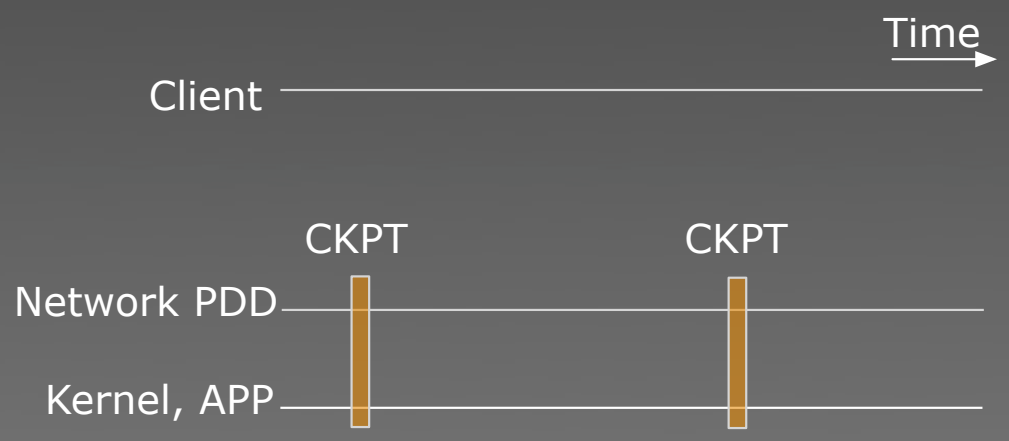


Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates



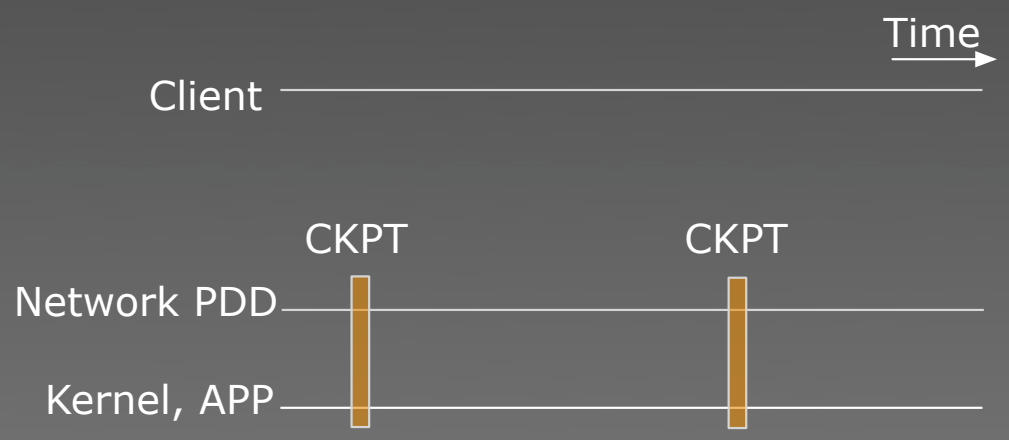
- Can avoid saving inputs replay
- Sender timeouts and resend



Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates

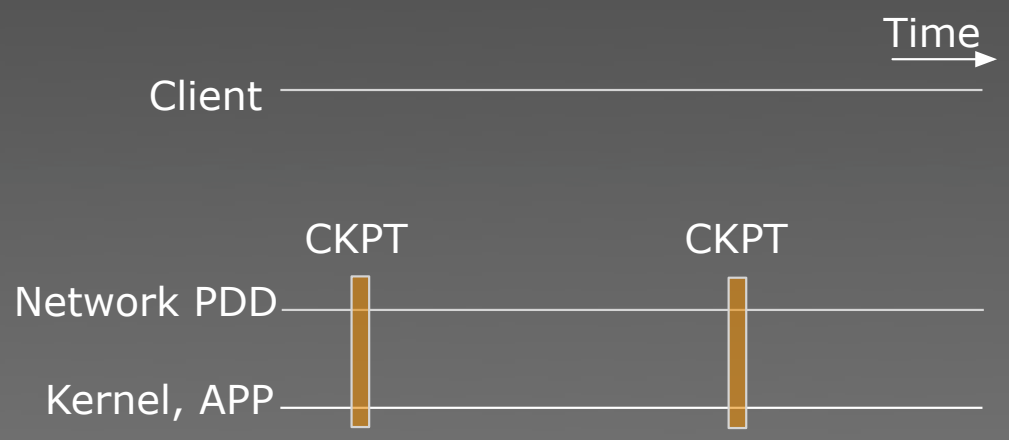
- Can avoid saving inputs replay
- Sender timeouts and resend



Network PDD and TCP

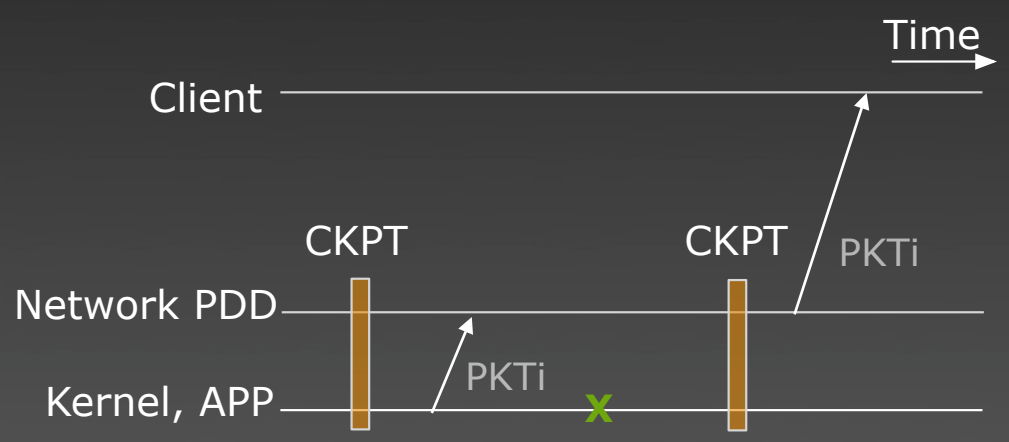
- Resend packets after rollback
- TCP eliminates duplicates

- Can avoid saving inputs replay
- Sender timeouts and resend

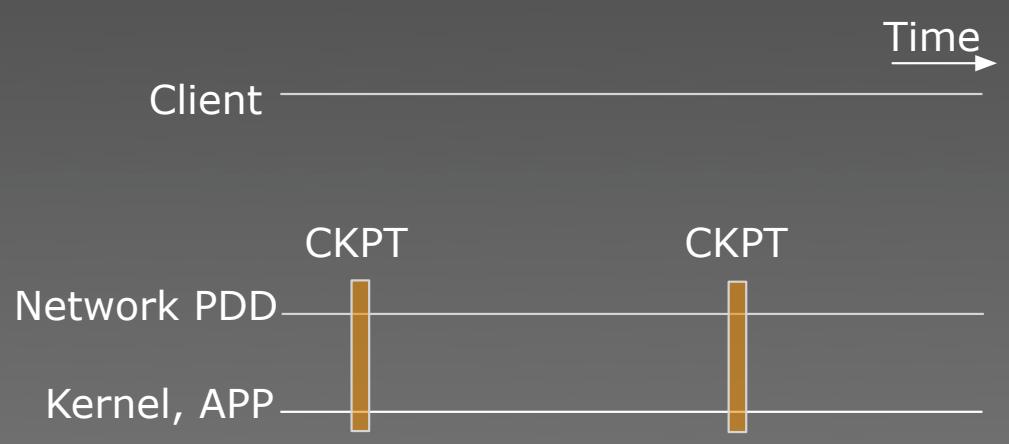


Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates

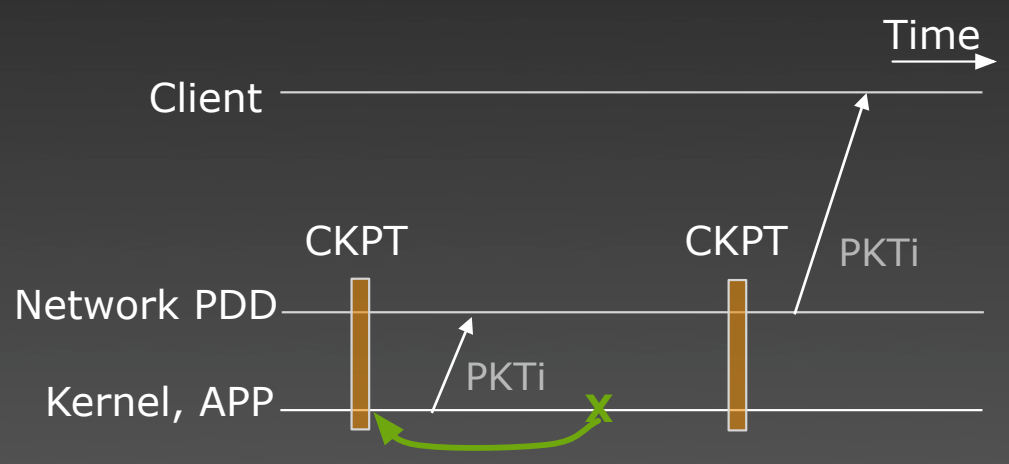


- Can avoid saving inputs replay
- Sender timeouts and resend

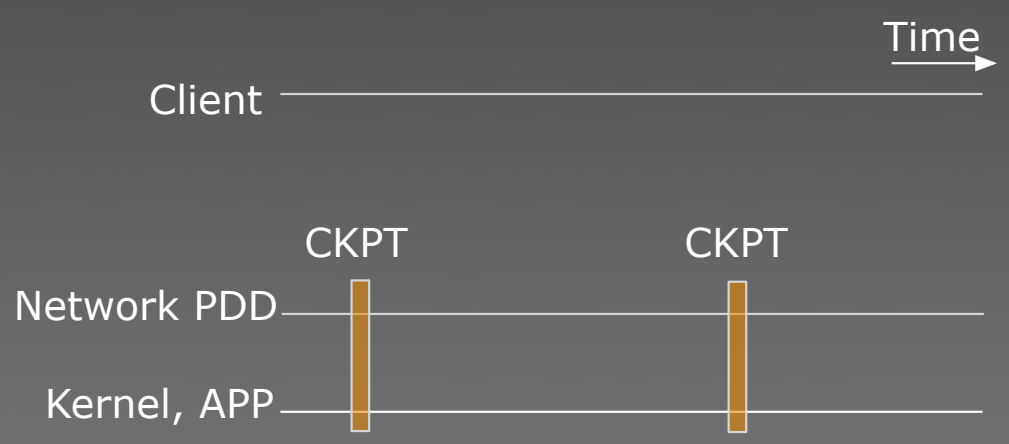


Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates

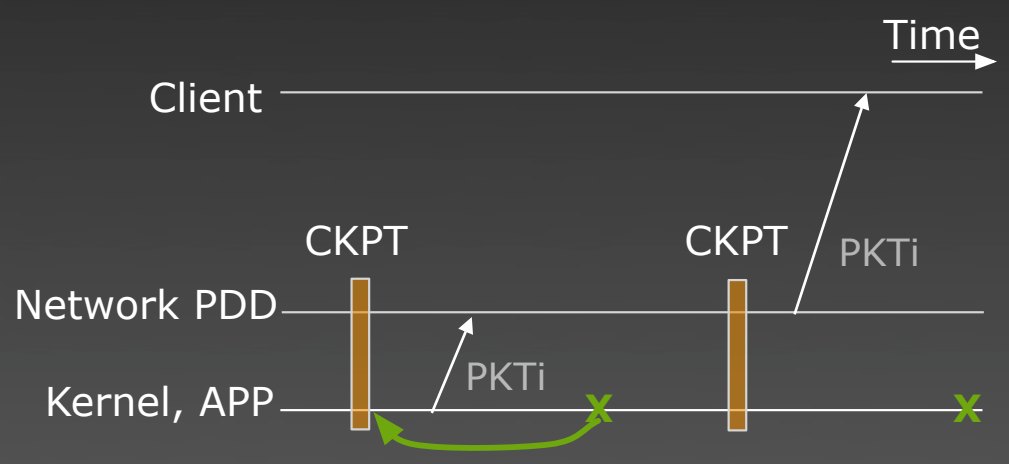


- Can avoid saving inputs replay
- Sender timeouts and resend

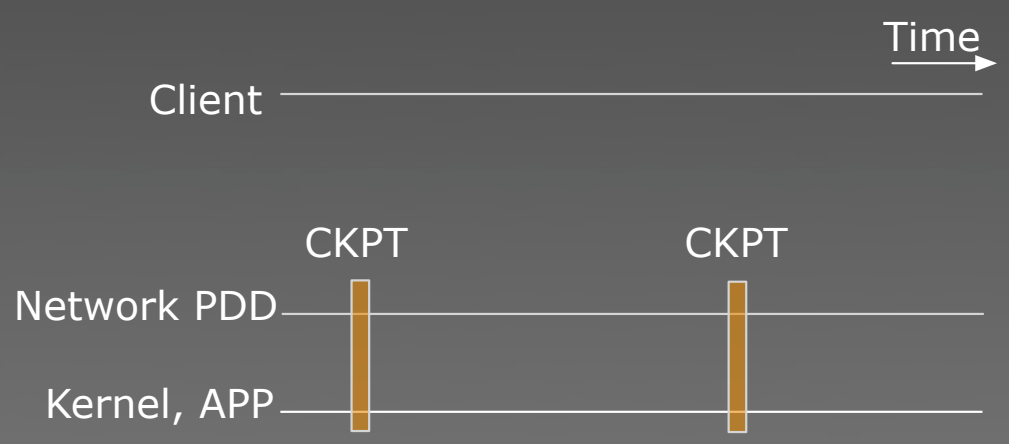


Network PDD and TCP

- Resend packets after rollback
- TCP eliminates duplicates

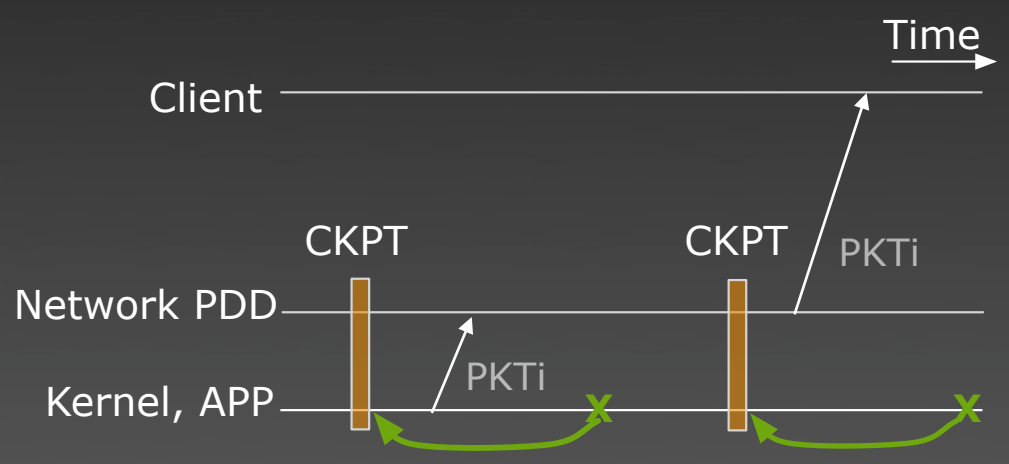


- Can avoid saving inputs replay
- Sender timeouts and resend

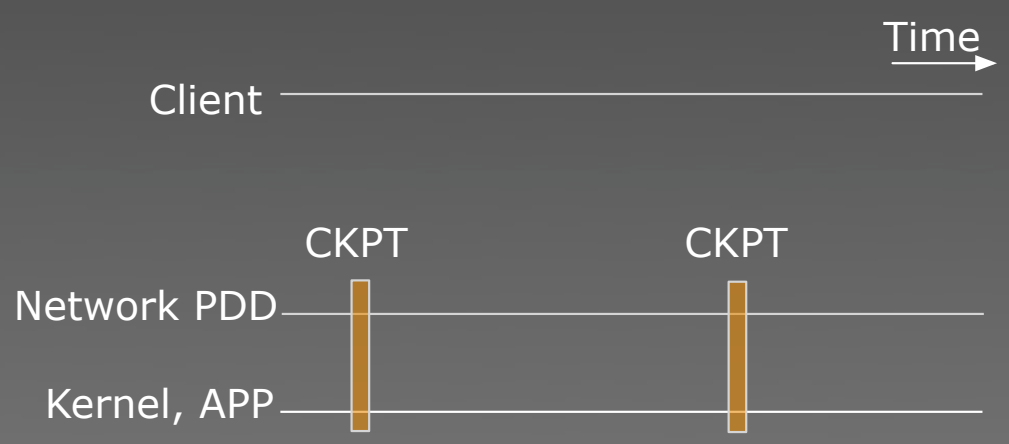


Network PDD and TCP

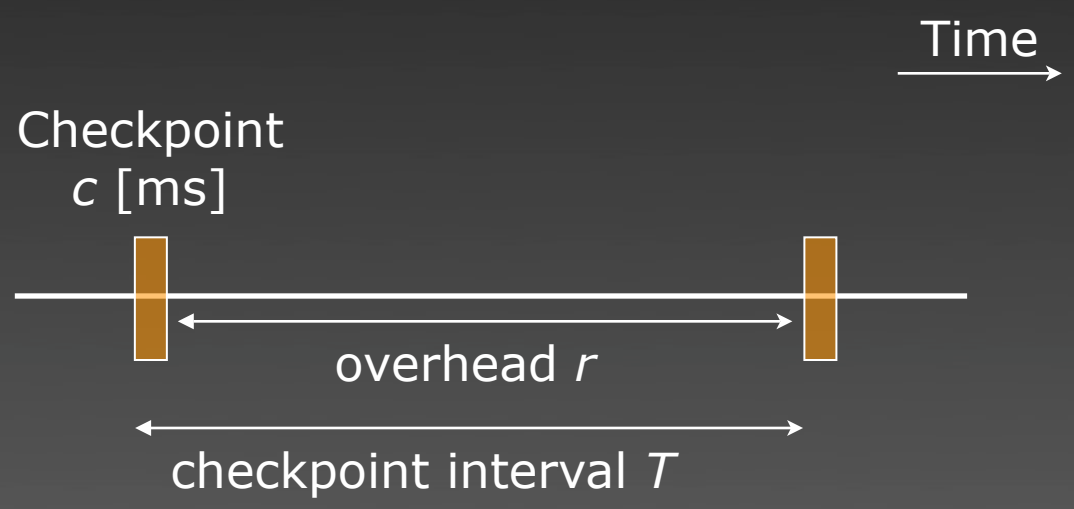
- Resend packets after rollback
- TCP eliminates duplicates



- Can avoid saving inputs replay
- Sender timeouts and resend

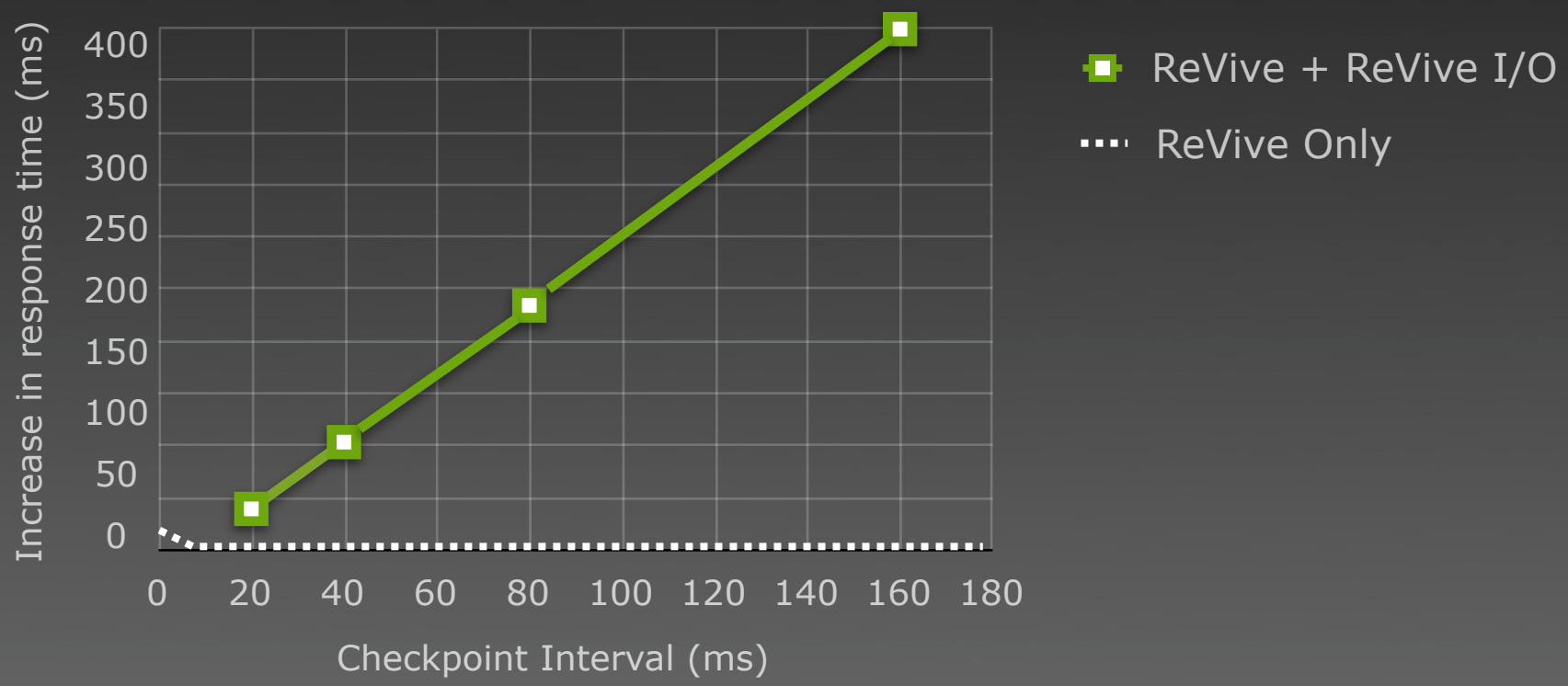


Integration with ReVive



- Simple model of ReVive overhead:
  - c and r are constants (regardless of checkpoint frequency)
  - For instance, c = 1 ms, r=5% and T = 100 ms give 6% overhead

Integration with ReVive (WebStone)



- Increase in a latency-bound workload is 2 x *T* (Checkpoint Interval)
- Most of the overhead comes from ReVive I/O