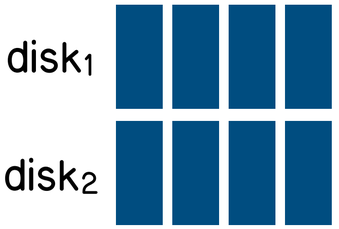
Argosy: Verifying layered storage systems with recovery refinement

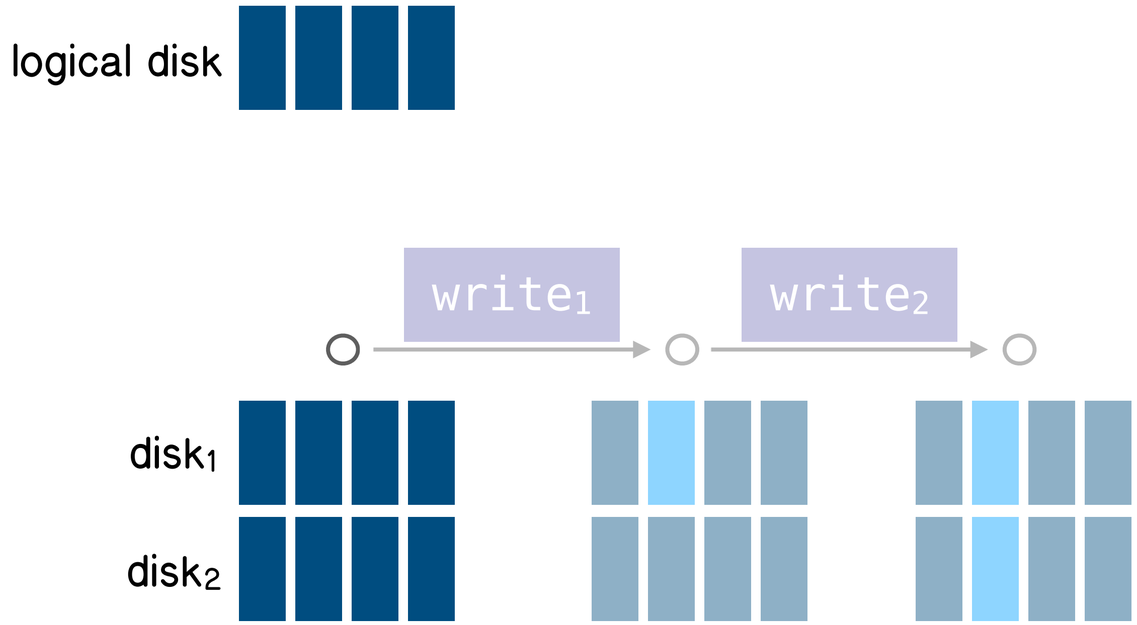
Tej Chajed, Joseph Tassarotti, Frans Kaashoek, Nickolai Zeldovich

MIT

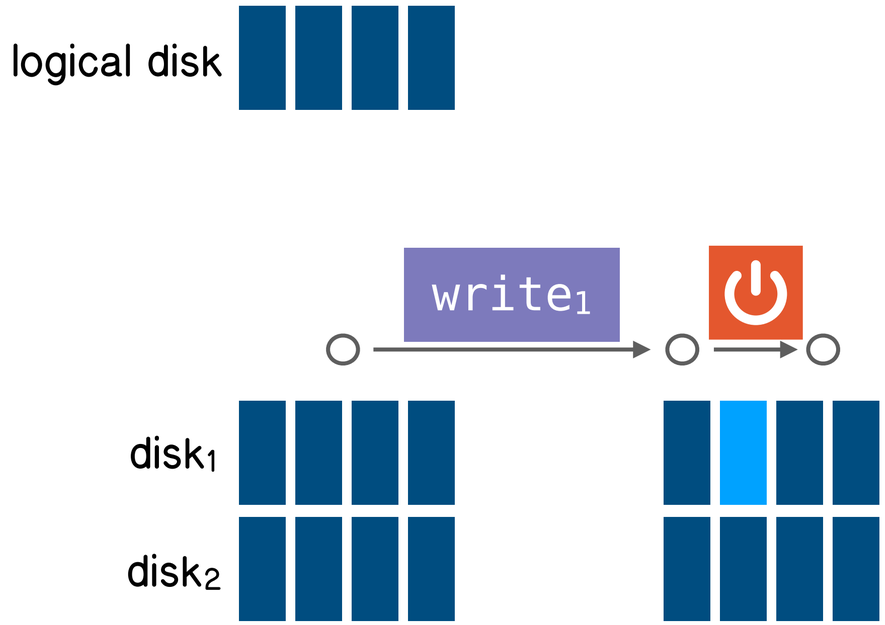
logical disk 



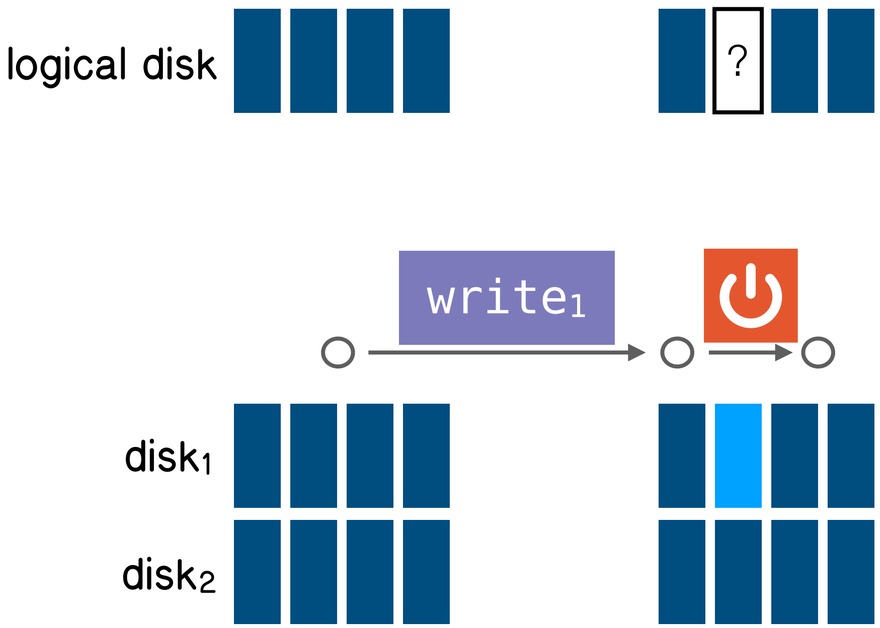
Bob writes a replication system



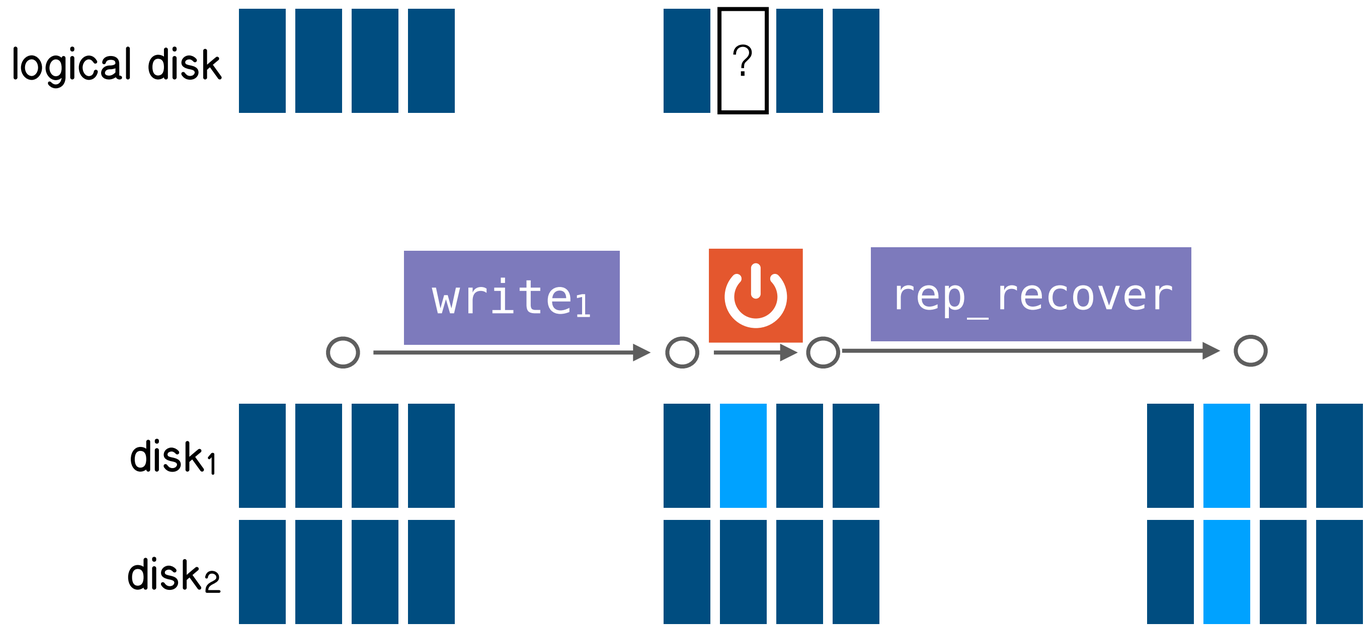
Bob writes a replication system



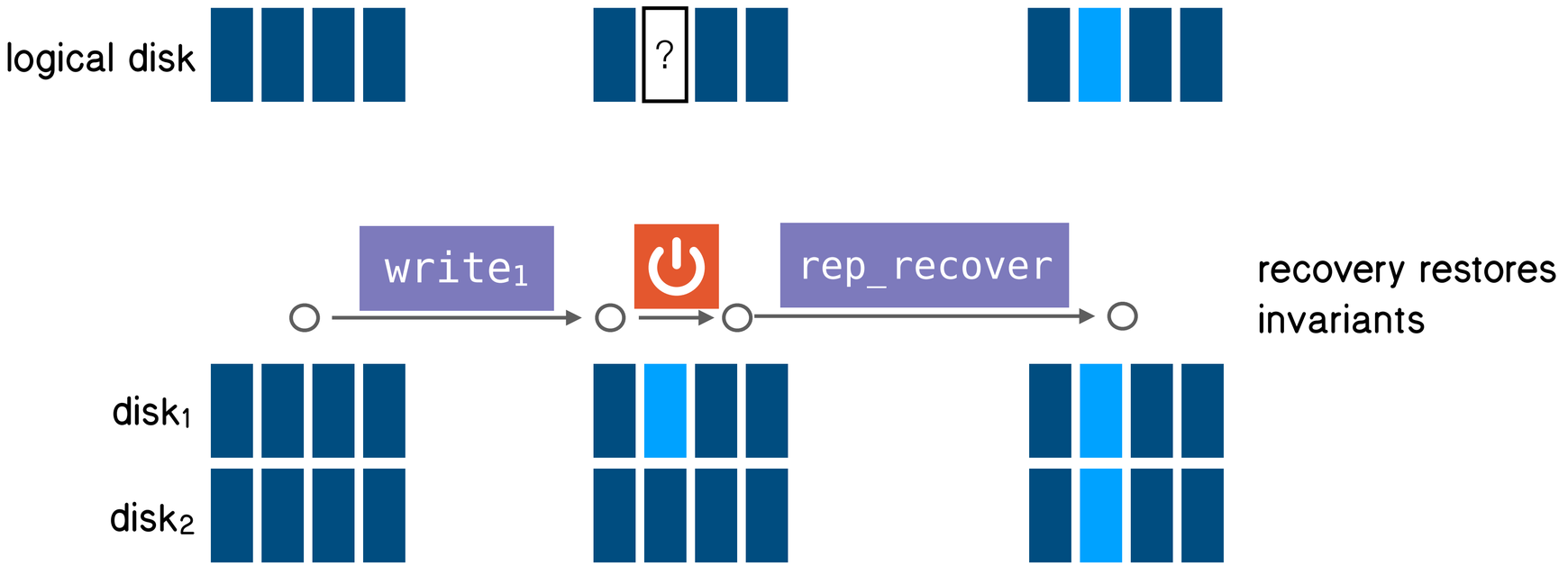
Bob writes a replication system



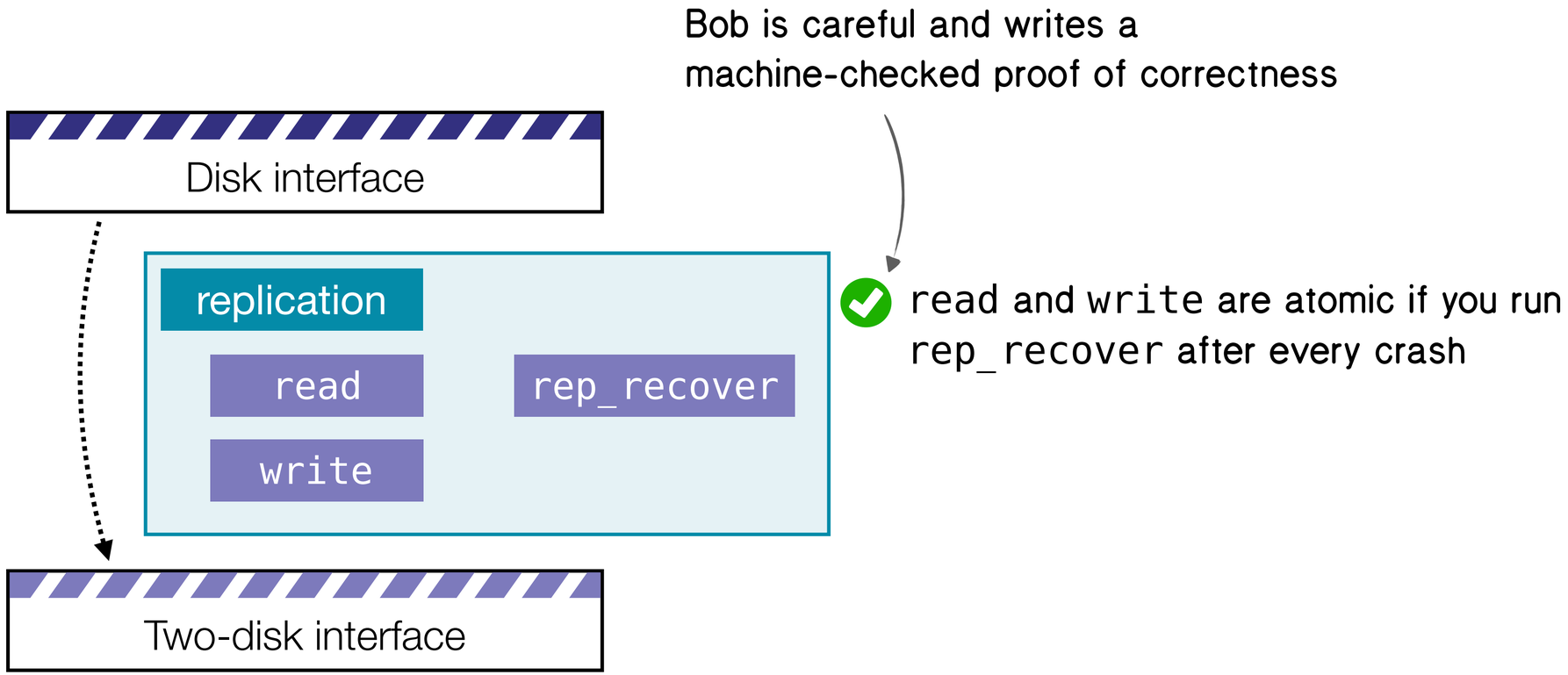
Bob writes a replication system

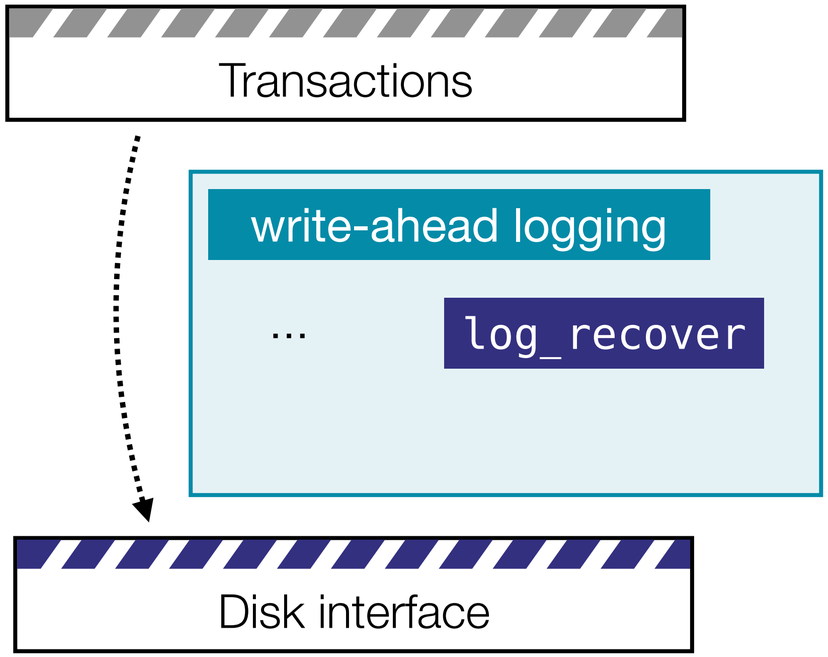


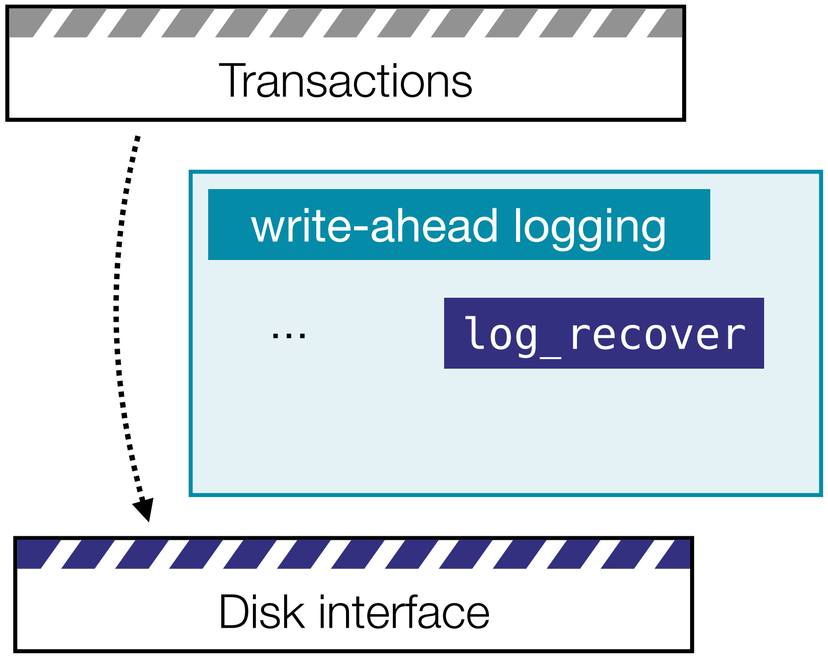
Bob writes a replication system and implements its recovery procedure



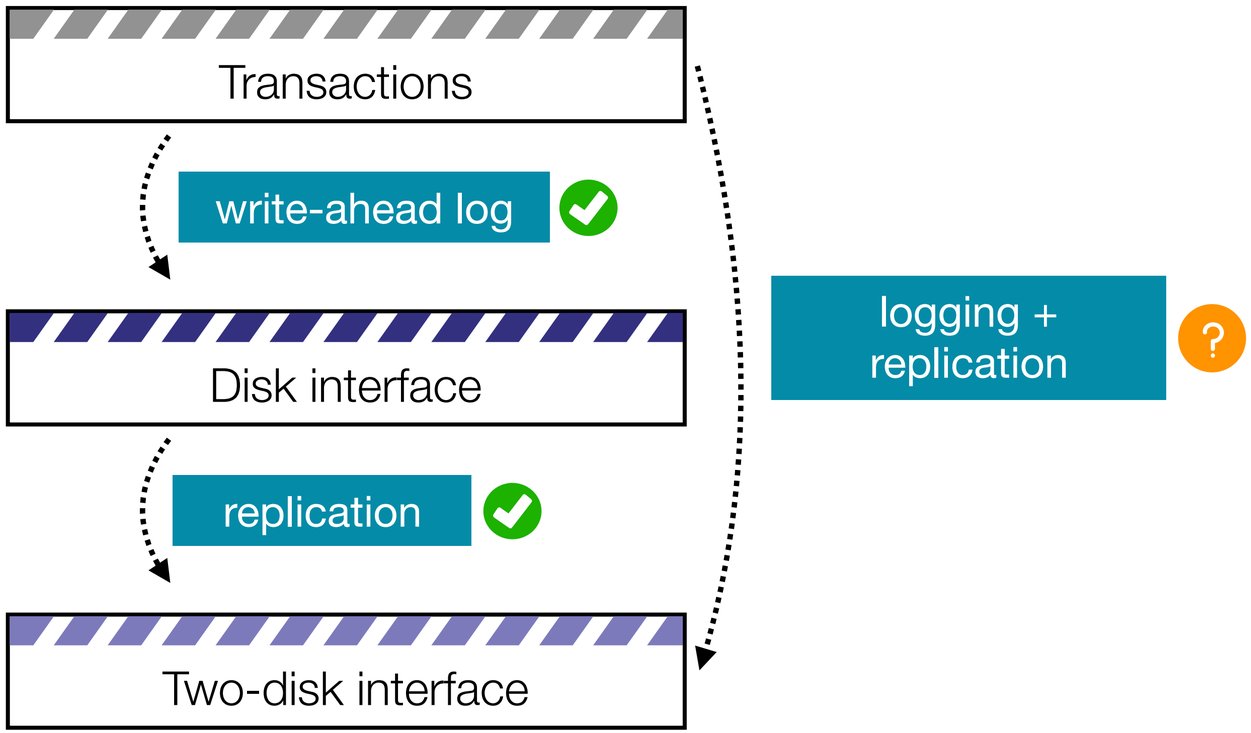
Bob writes a replication system and implements its recovery procedure

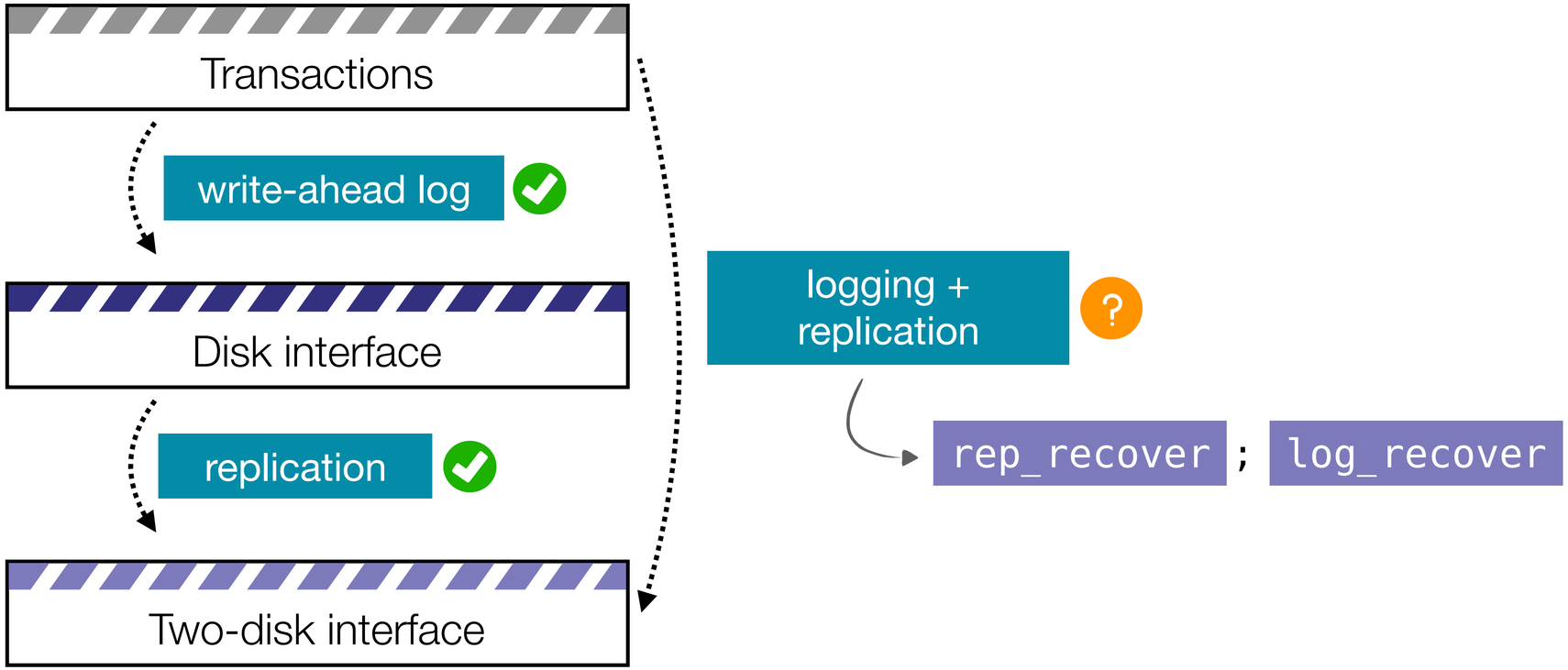

Bob is careful and writes a
machine-checked proof of correctness
Disk interface
replication
read
rep_recover
write
Two-disk interface
read and write are atomic if you run
rep_recover after every crash


Transactions
write-ahead logging
...
log_recover
Disk interface

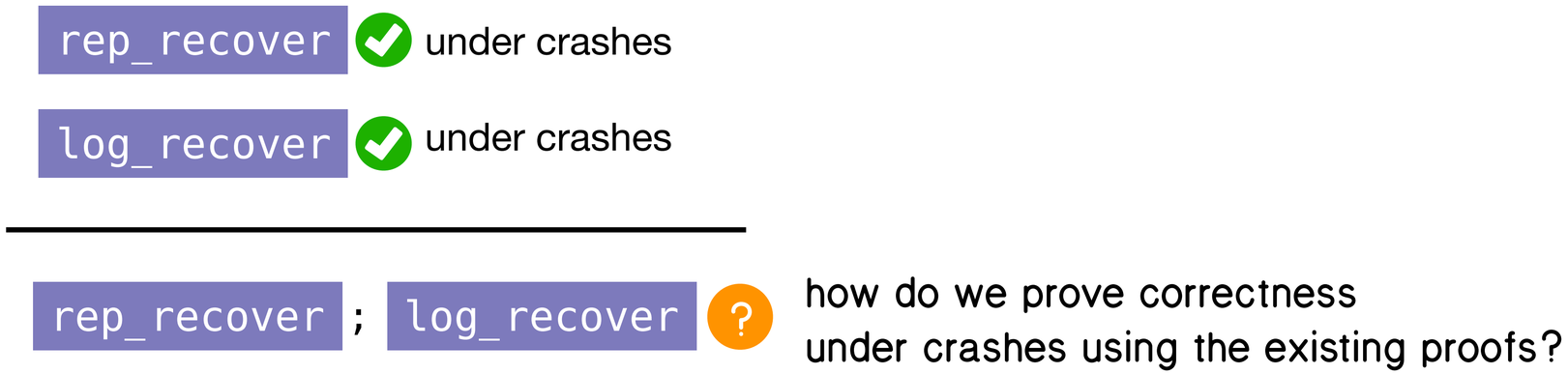


ops are atomic if you run `log_recover` after every crash

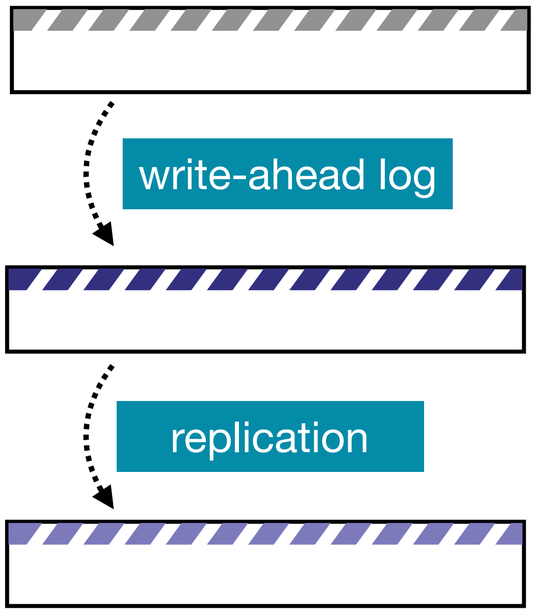

Transactions
write-ahead log
Disk interface
replication
Two-disk interface
logging +
replication
?


Transactions
write-ahead log
Disk interface
replication
Two-disk interface
logging +
replication
?
rep_recover ; log_recover

Challenge: crashes during composed recovery



Prior work cannot handle multiple recovery procedures

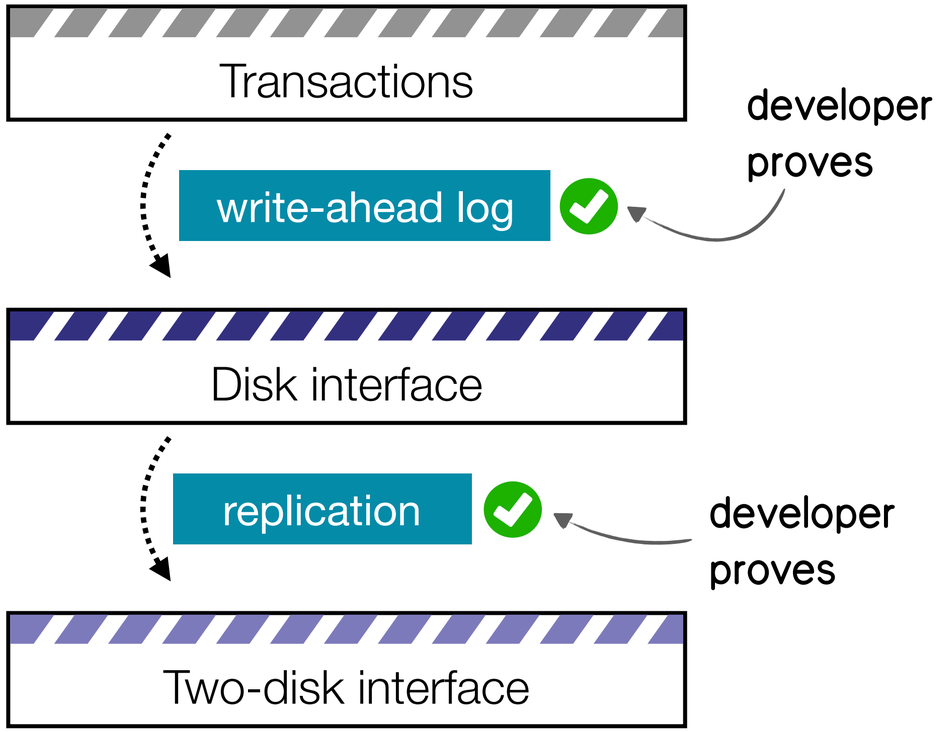


CHL [SOSP ’15] — not modular

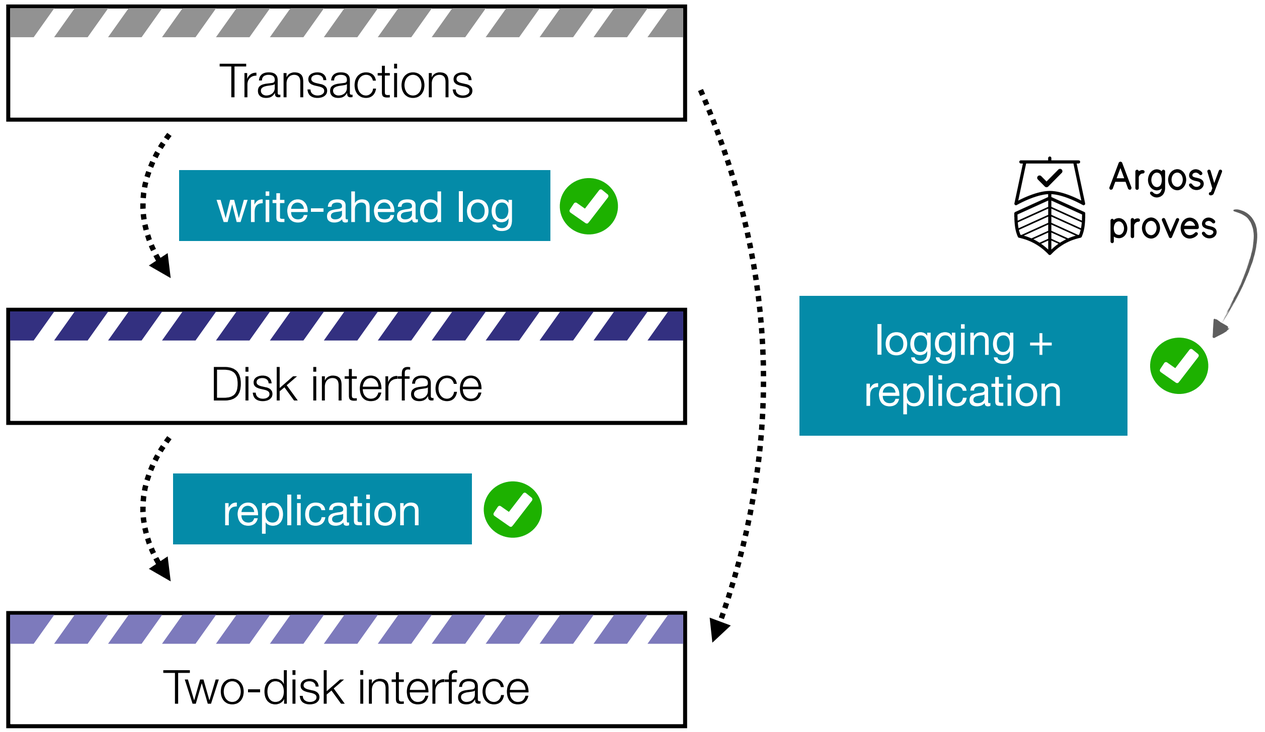
Yggdrasil [OSDI ’16] — single recovery

Flashix [SCP ’16] — restricted recovery procedures

Argosy supports modular recovery proofs



Argosy supports modular recovery proofs



Contributions

Recovery refinement for modular proofs

Contributions

Recovery refinement for modular proofs

see paper CHL for proving recovery refinement

see paper Verified example: logging + replication

Contributions

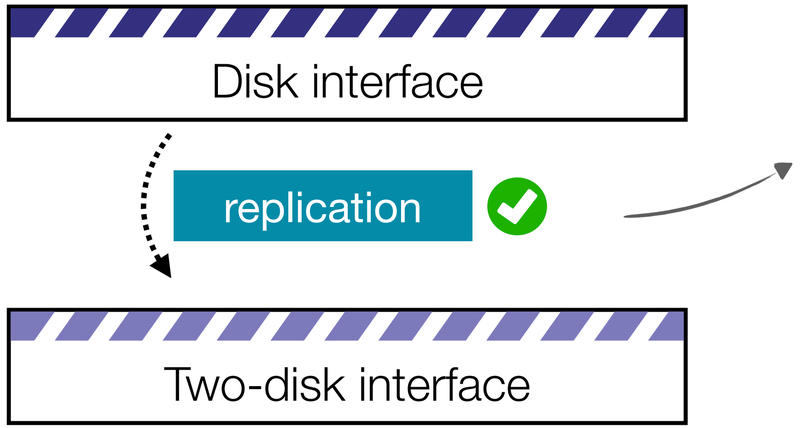
Recovery refinement for modular proofs

see paper — CHL for proving recovery refinement

see paper — Verified example: logging + replication

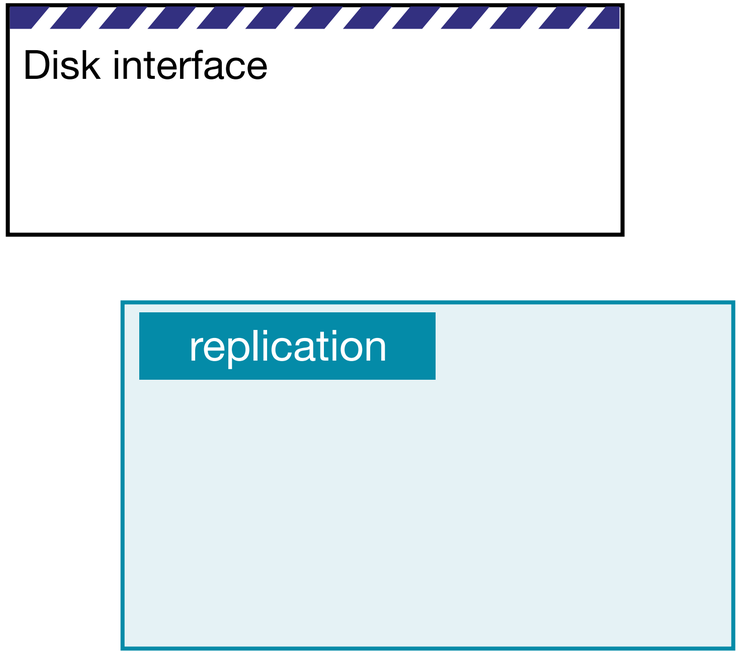
see code — Machine-checked proofs in Coq 

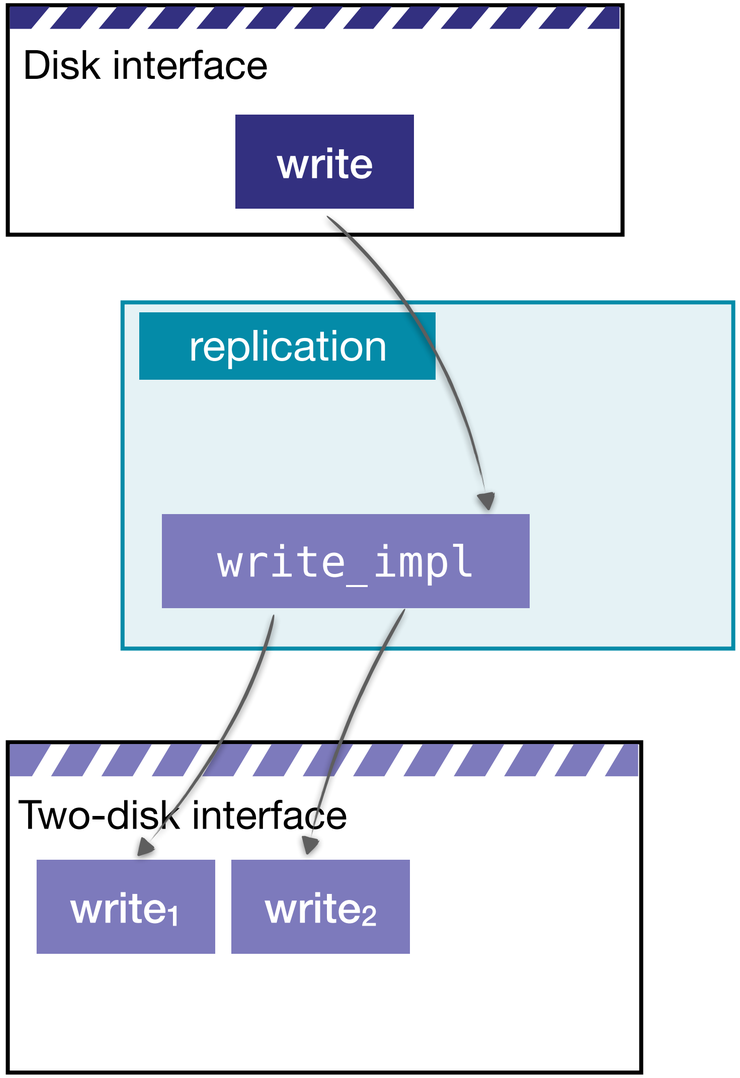
Preview: recovery refinement

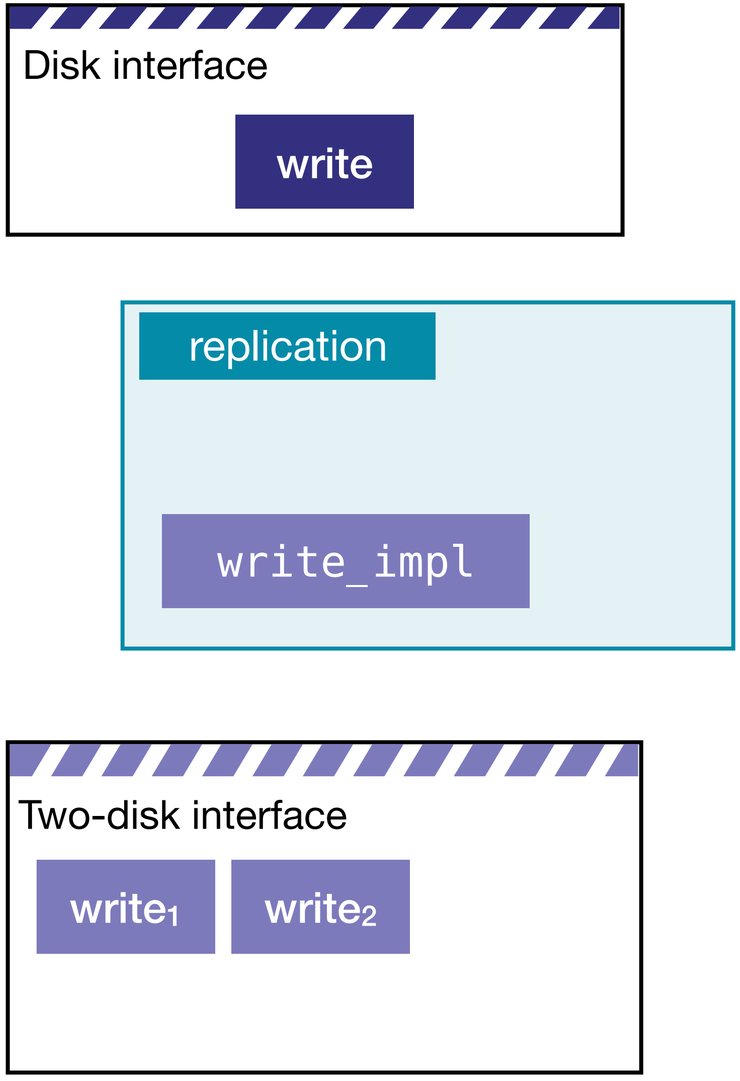


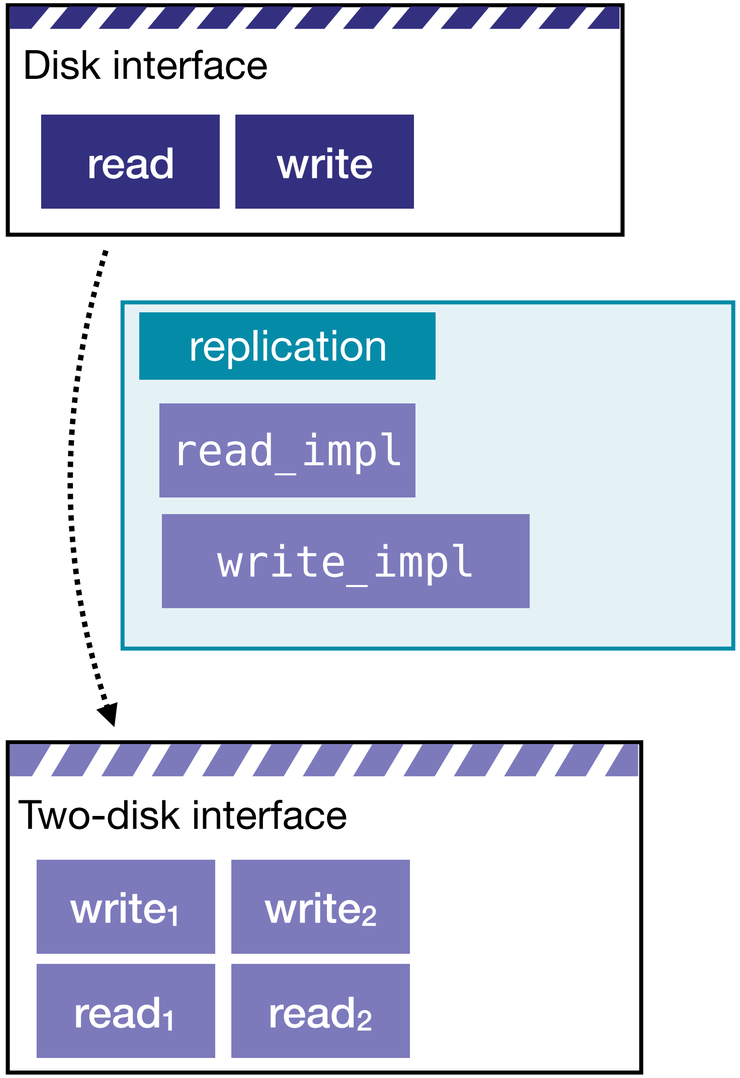
1. Normal execution correctness using *refinement*
2. Crash and recovery correctness using *recovery refinement*

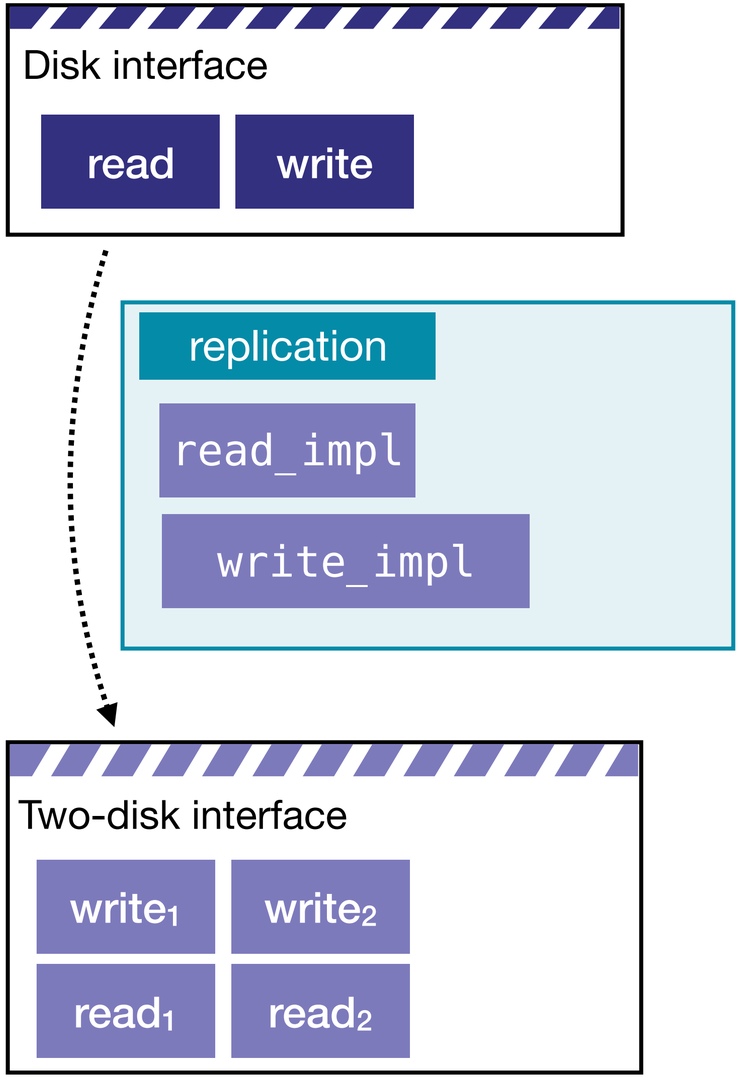
Refinement


Disk interface
replication
Two-disk interface

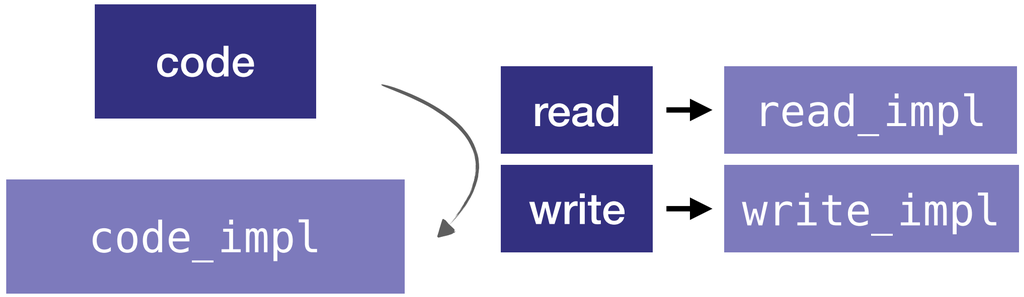

Disk interface
write
replication
write_impl
Two-disk interface
write1
write2


Disk interface
write
replication
write_impl
Two-disk interface
write1
write2

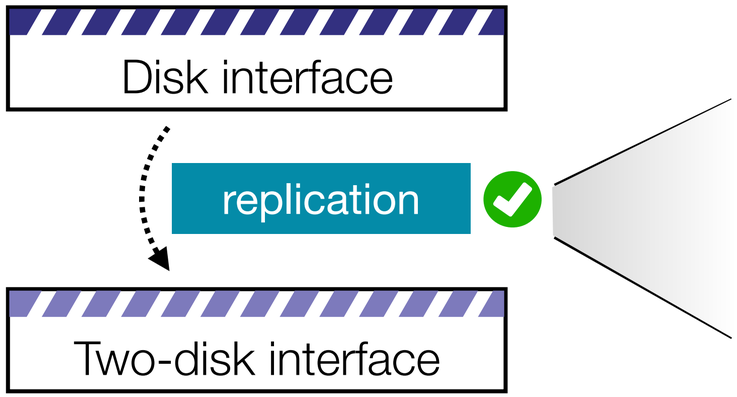

Disk interface
read
write
replication
read_impl
write_impl
Two-disk interface
write1
write2
read1
read2

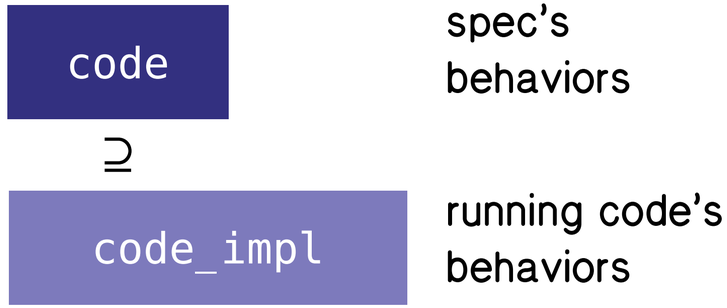


correctness is based on how we use replication:
run code using Disk interface on top of two disks



Correctness: trace inclusion

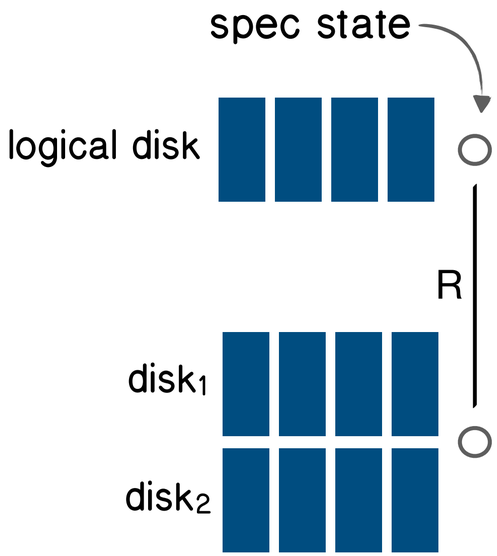




spec's behaviors

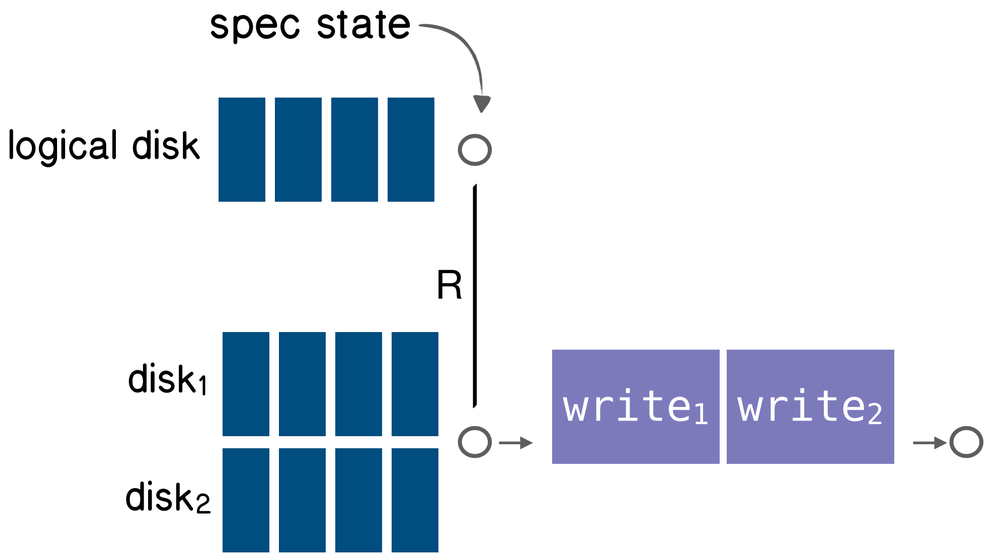
running code's behaviors

Proving correctness with an abstraction relation



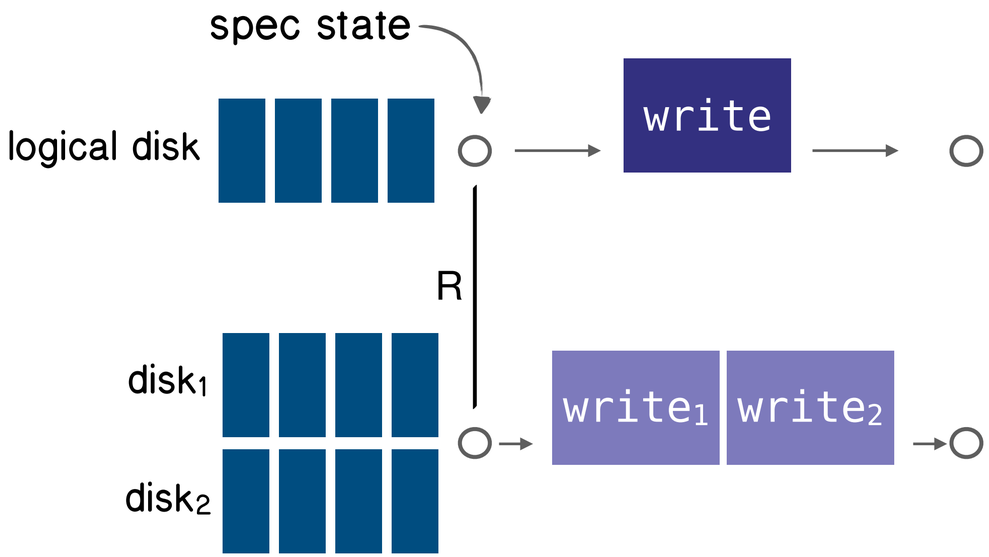
1. developer provides abstraction relation R

Proving correctness with an abstraction relation



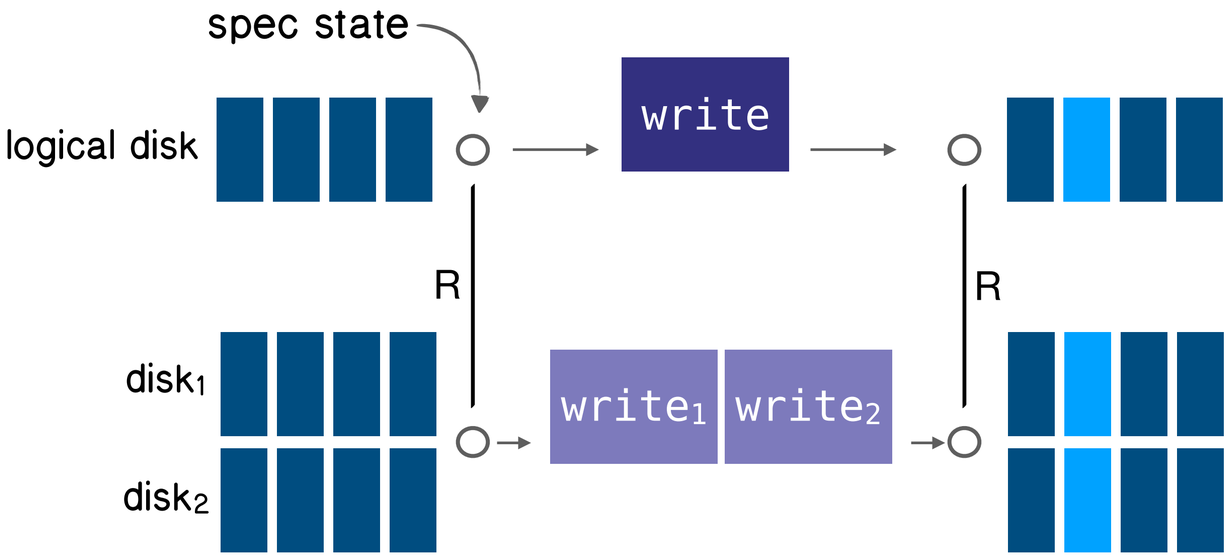
1. developer provides abstraction relation R

Proving correctness with an abstraction relation



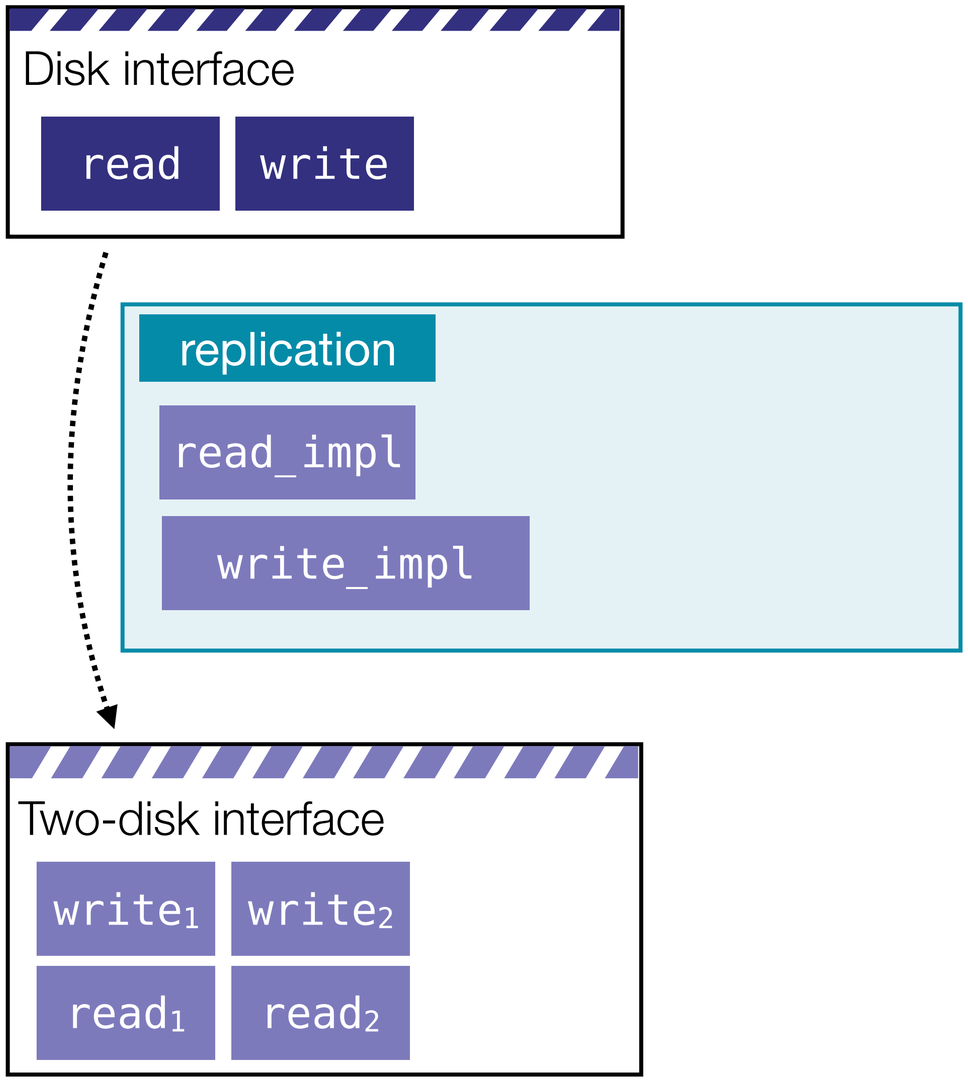
1. developer provides abstraction relation R
2. prove spec execution exists

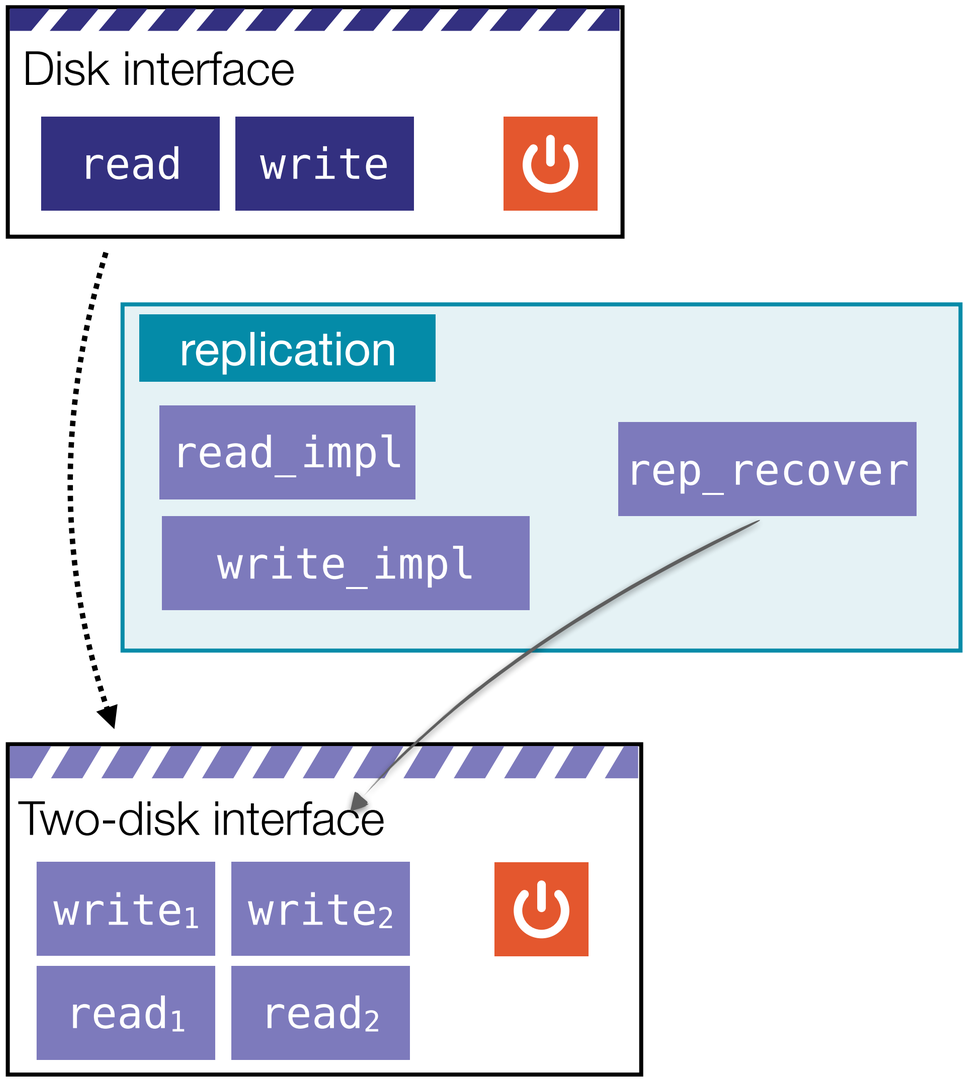
Proving correctness with an abstraction relation

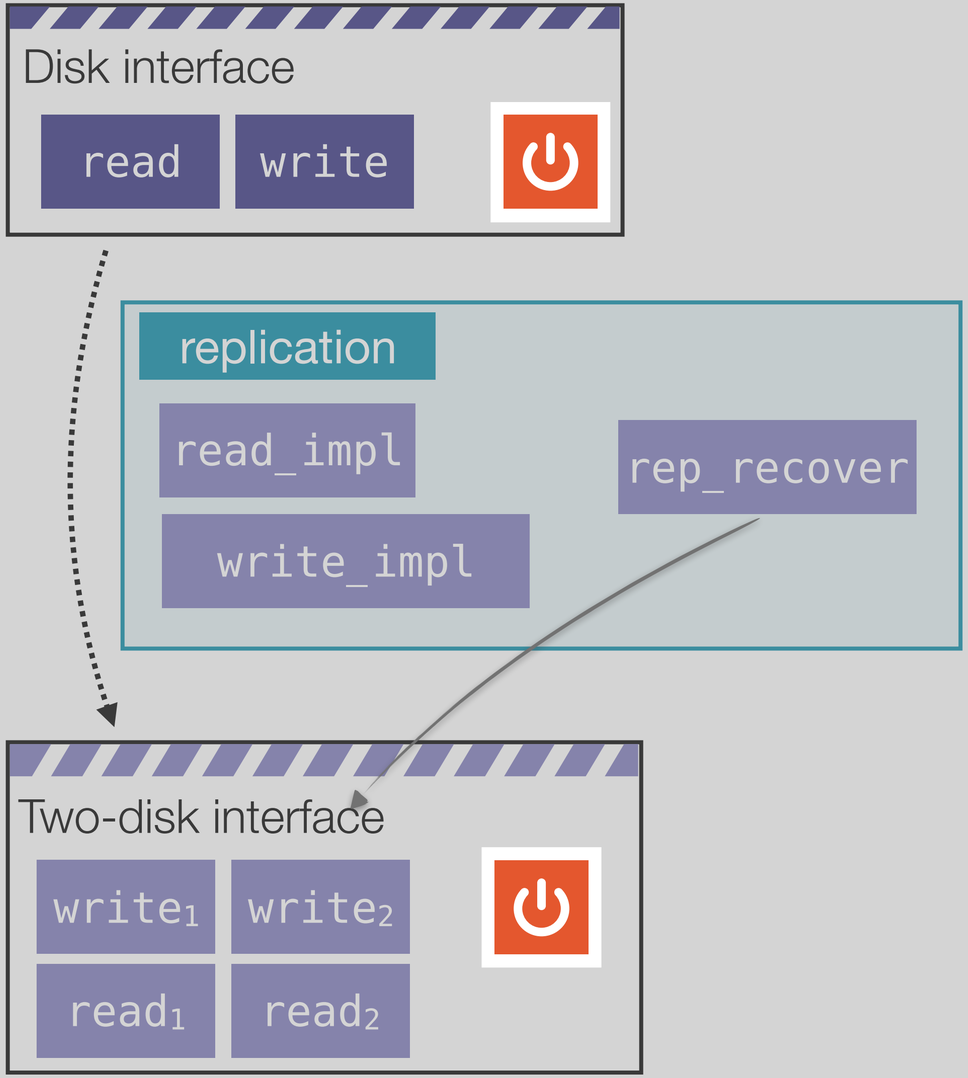


1. developer provides abstraction relation R
2. prove spec execution exists
3. and abstraction relation is preserved

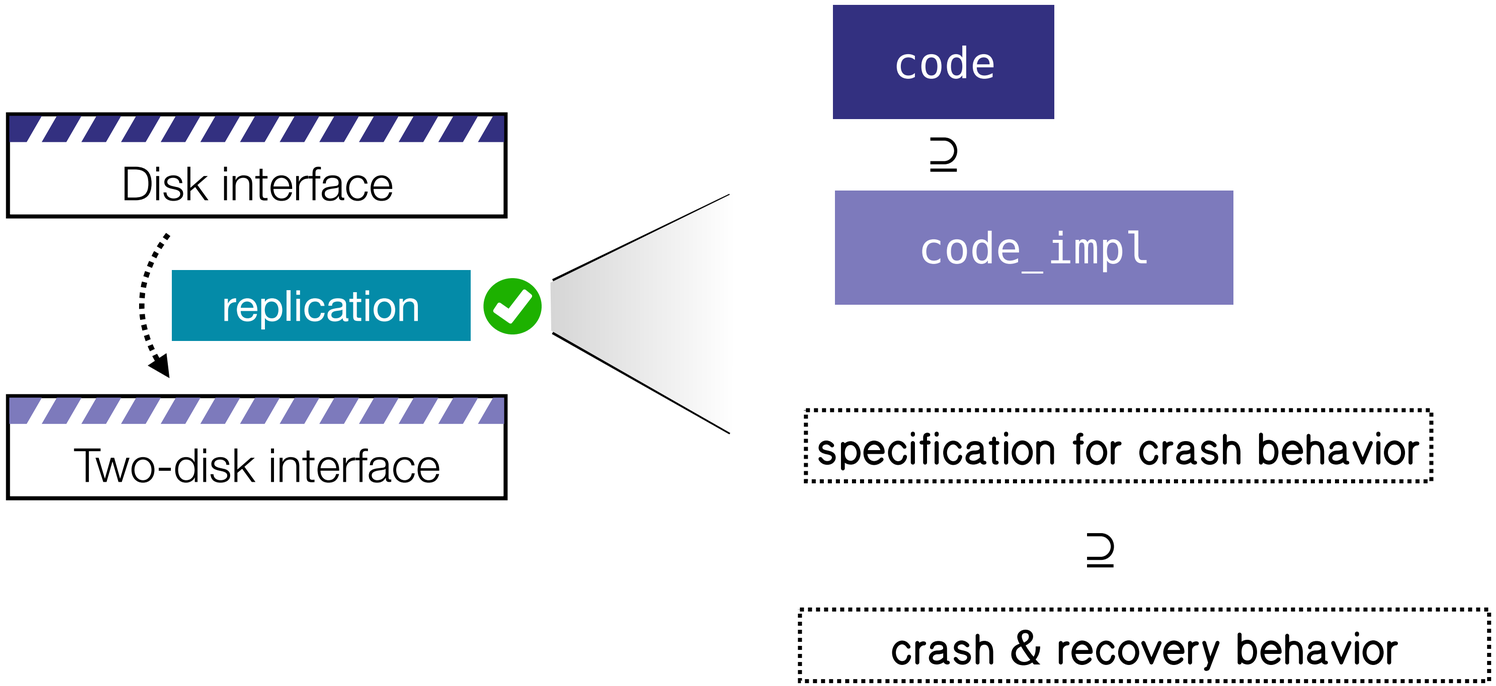
Recovery refinement


Disk interface
read
write
replication
read_impl
write_impl
Two-disk interface
write1
write2
read1
read2

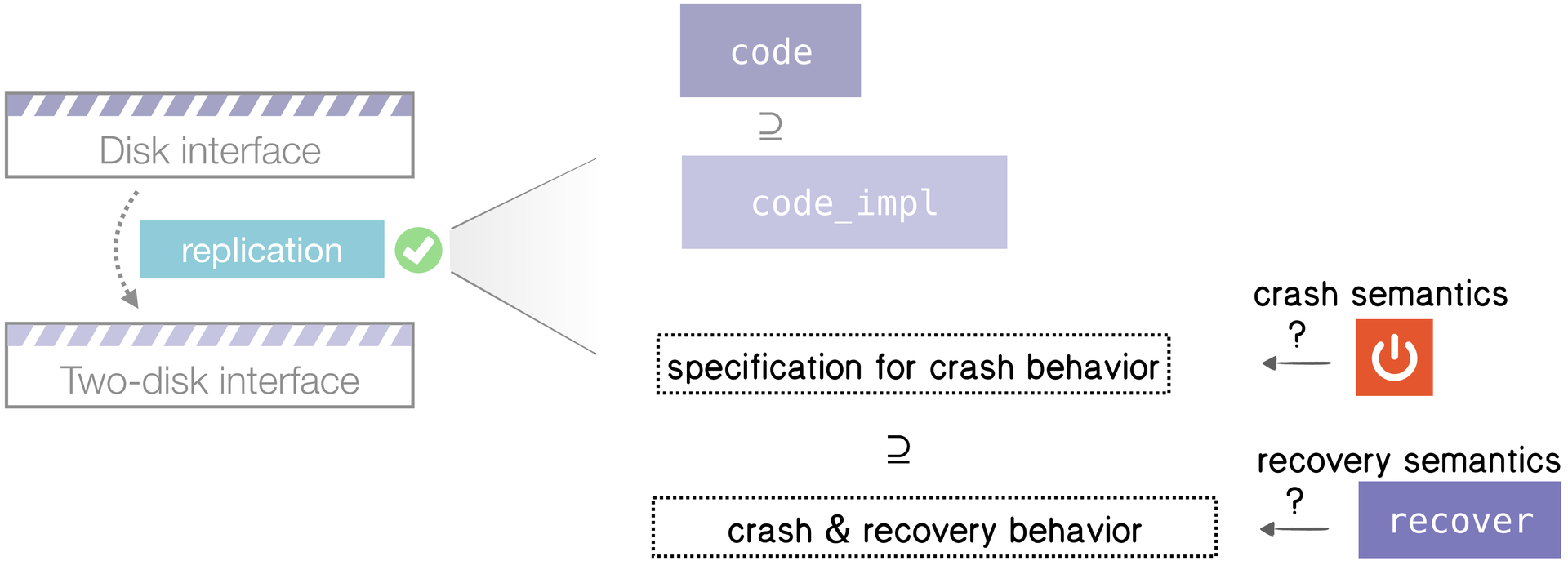

Disk interface
read
write
replication
read_impl
write_impl
rep_recover
Two-disk interface
write1
write2
read1
read2

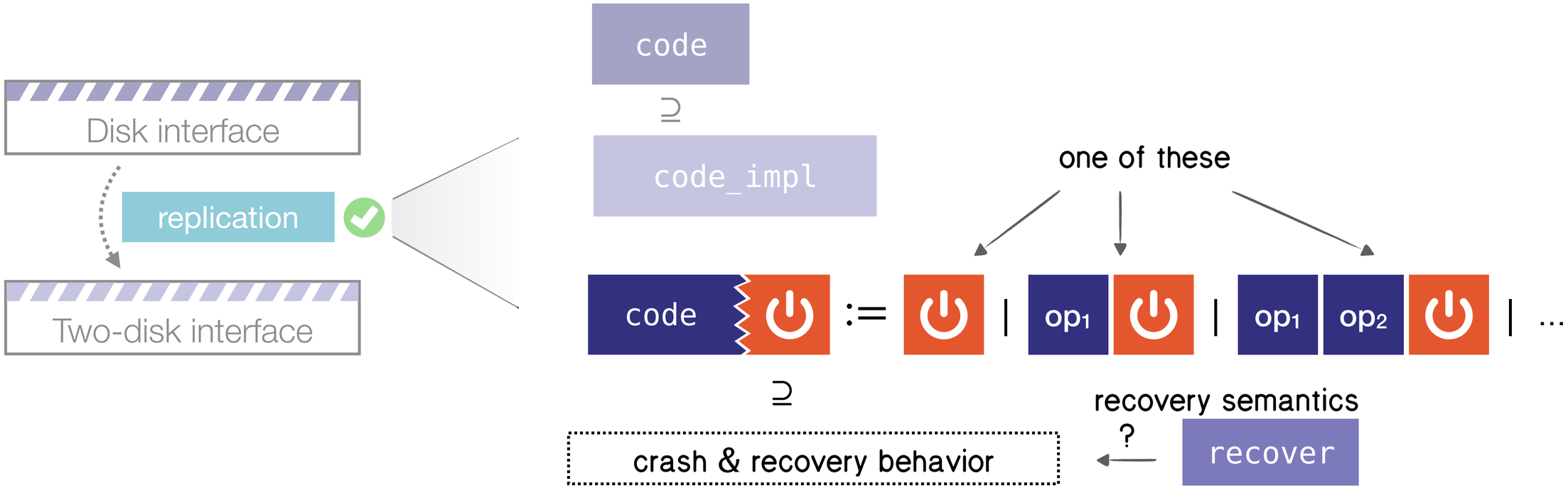

Disk interface
read
write
replication
read_impl
write_impl
rep_recover
Two-disk interface
write1
write2
read1
read2

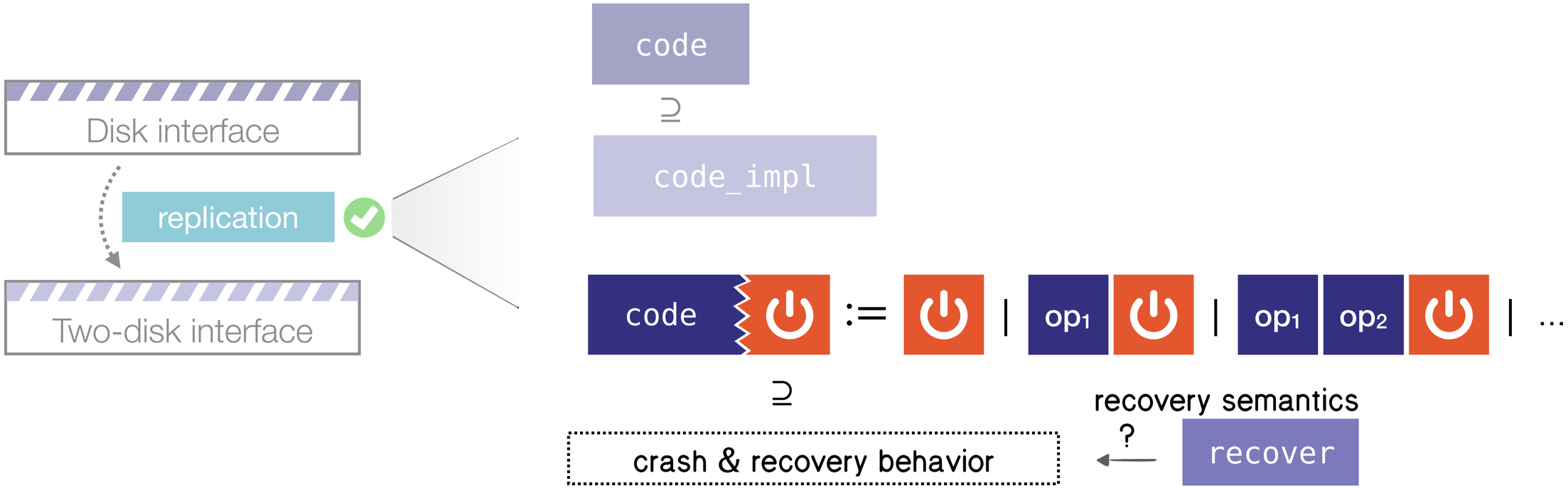
Extending trace inclusion with recovery

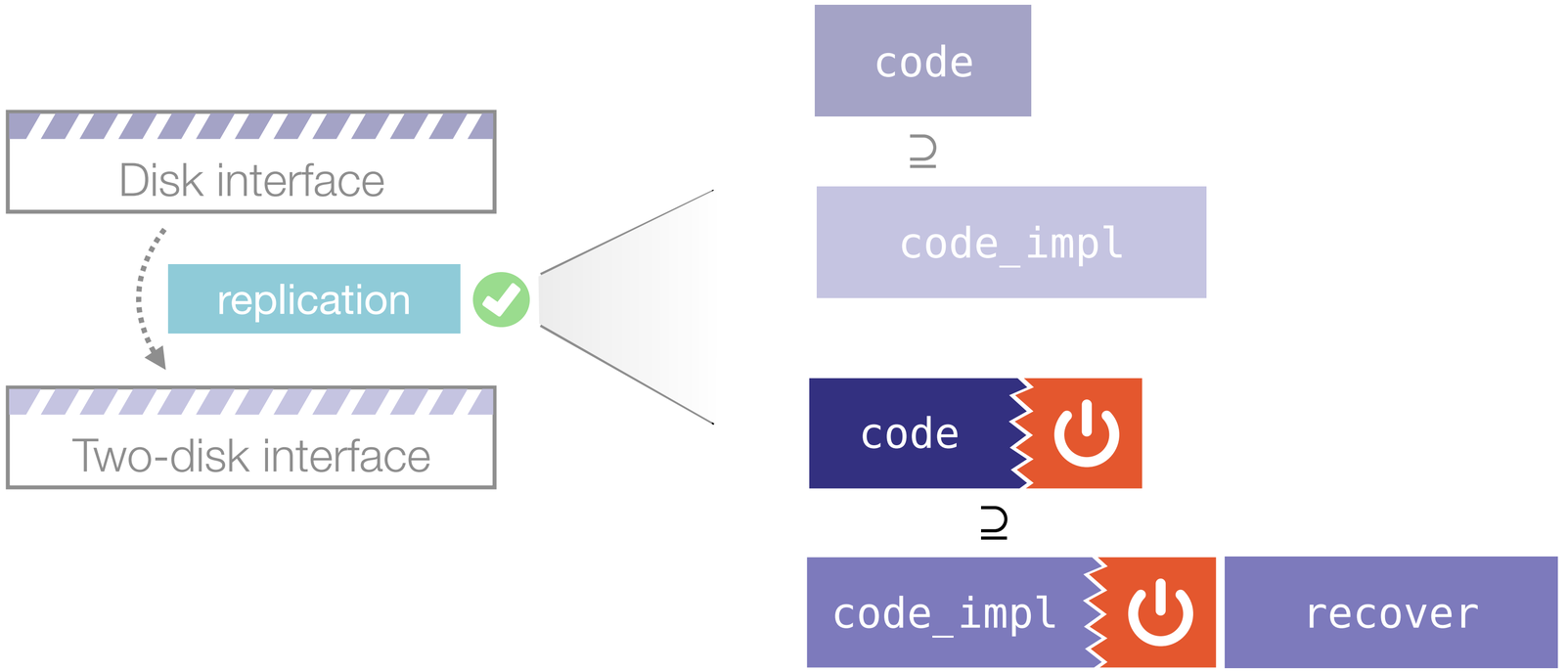


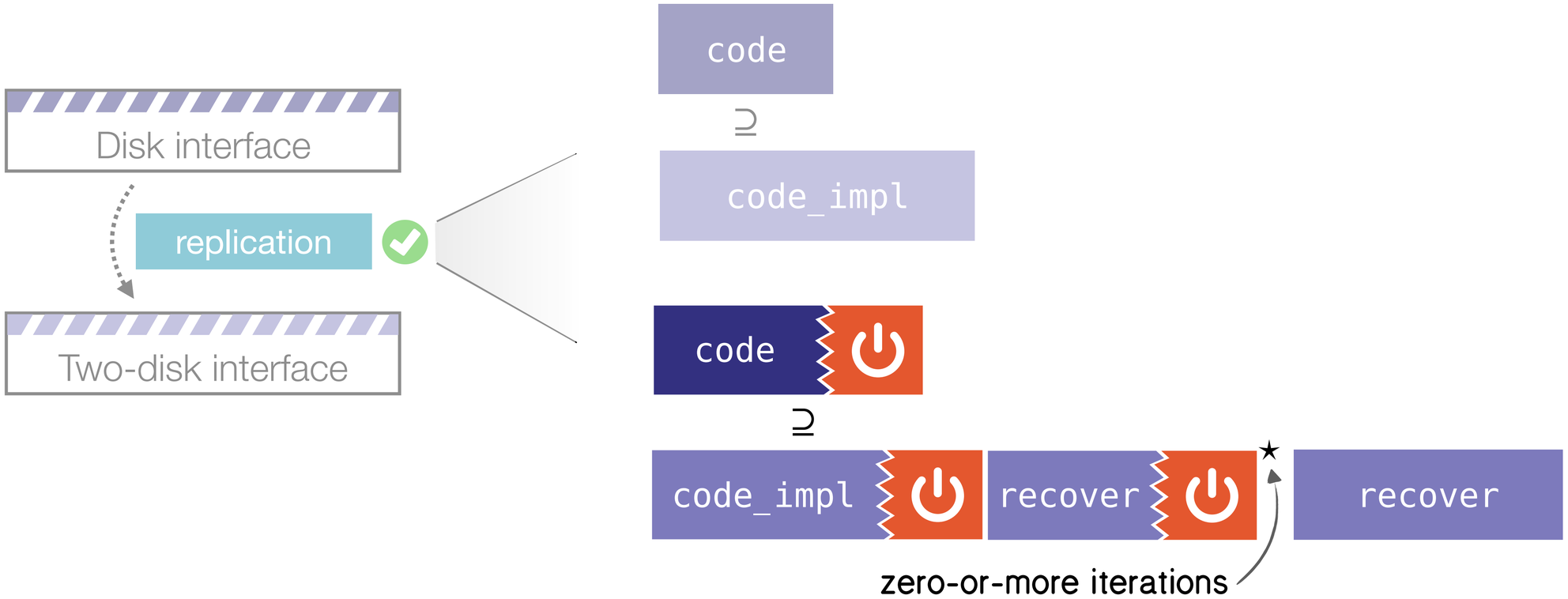
Extending trace inclusion with recovery

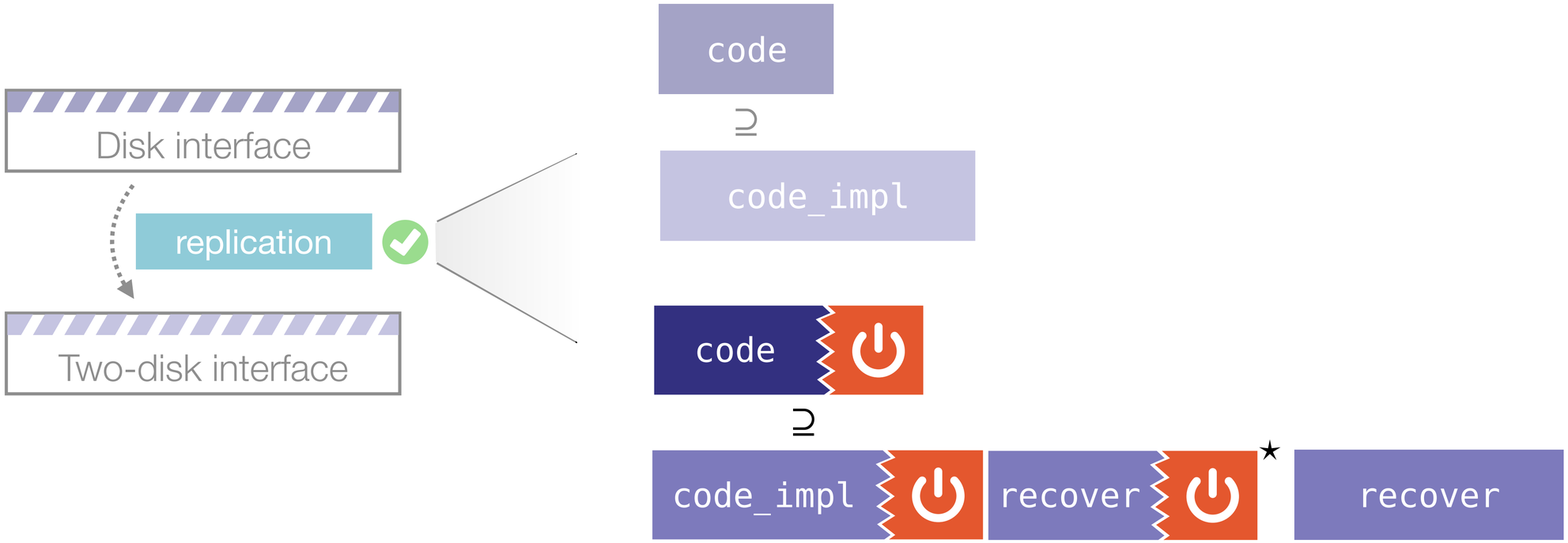



Disk interface
replication
Two-disk interface
code
⊇
code_impl
one of these
code
:=
op1
op1
op2
...
⊇
crash & recovery behavior
recovery semantics
?
recover

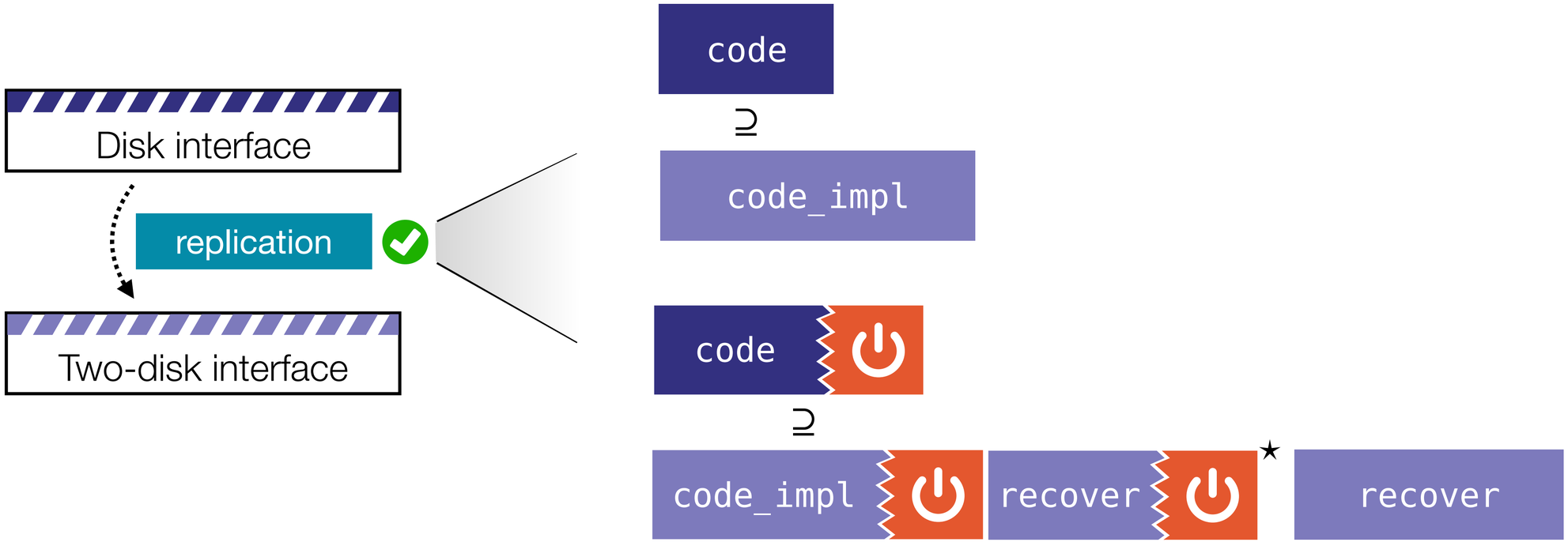

Disk interface
replication
Two-disk interface
code
⊇
code_impl
code
:=
op1
op1
op2
...
⊇
crash & recovery behavior
recovery semantics
?
recover


Disk interface
replication
Two-disk interface
code
⊇
code_impl
code
⊇
code_impl
recover


Disk interface
replication
Two-disk interface
code
⊇
code_impl
code
⊇
code_impl
recover
recover
★
zero-or-more iterations

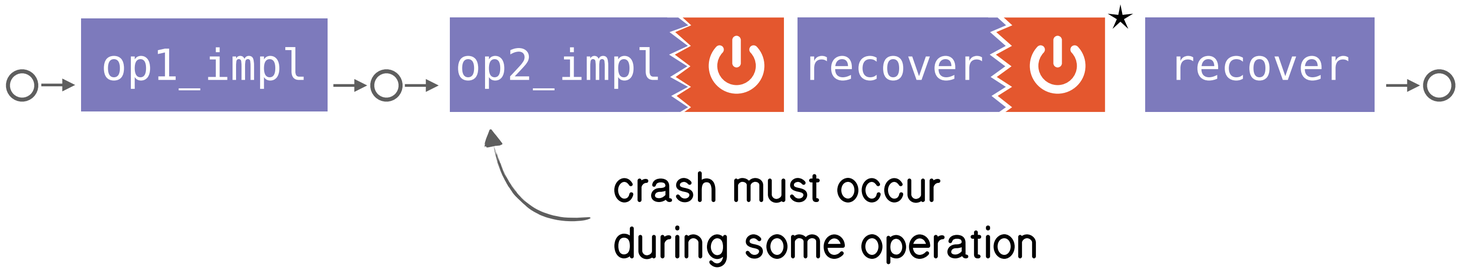

Disk interface
replication
Two-disk interface
code
⊇
code_impl
code
⊇
code_impl
recover
★
recover

Trace inclusion, with recovery



Proving trace inclusion, with recovery

Proving trace inclusion, with recovery



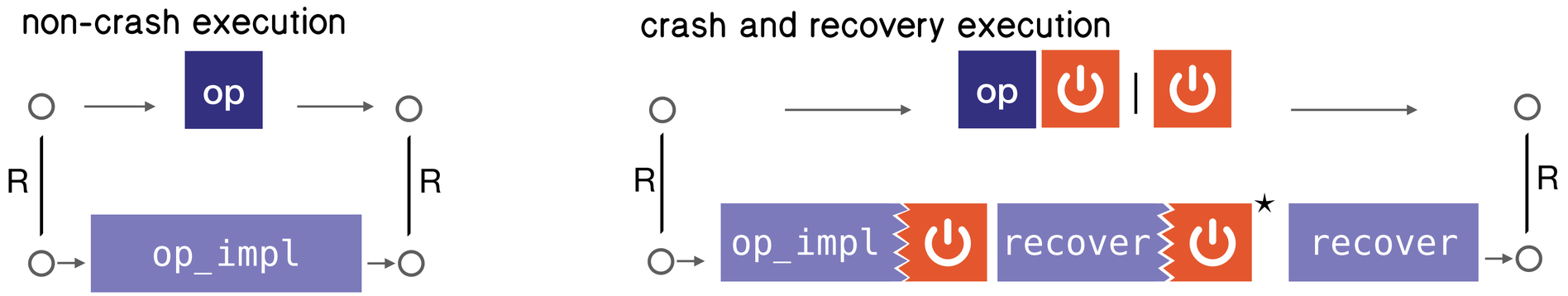
Proving trace inclusion, with recovery

Proving trace inclusion, with recovery

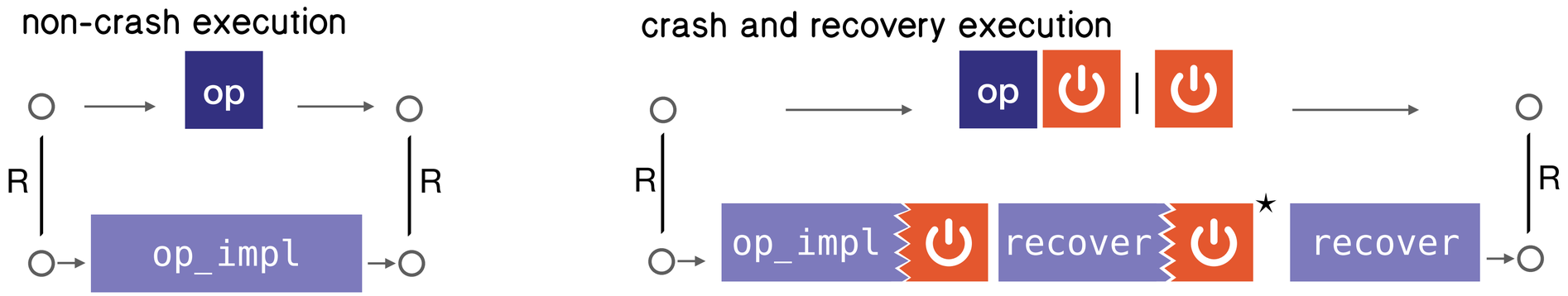
Proving trace inclusion, with recovery

Proving trace inclusion, with recovery

Recovery refinement



Recovery refinement



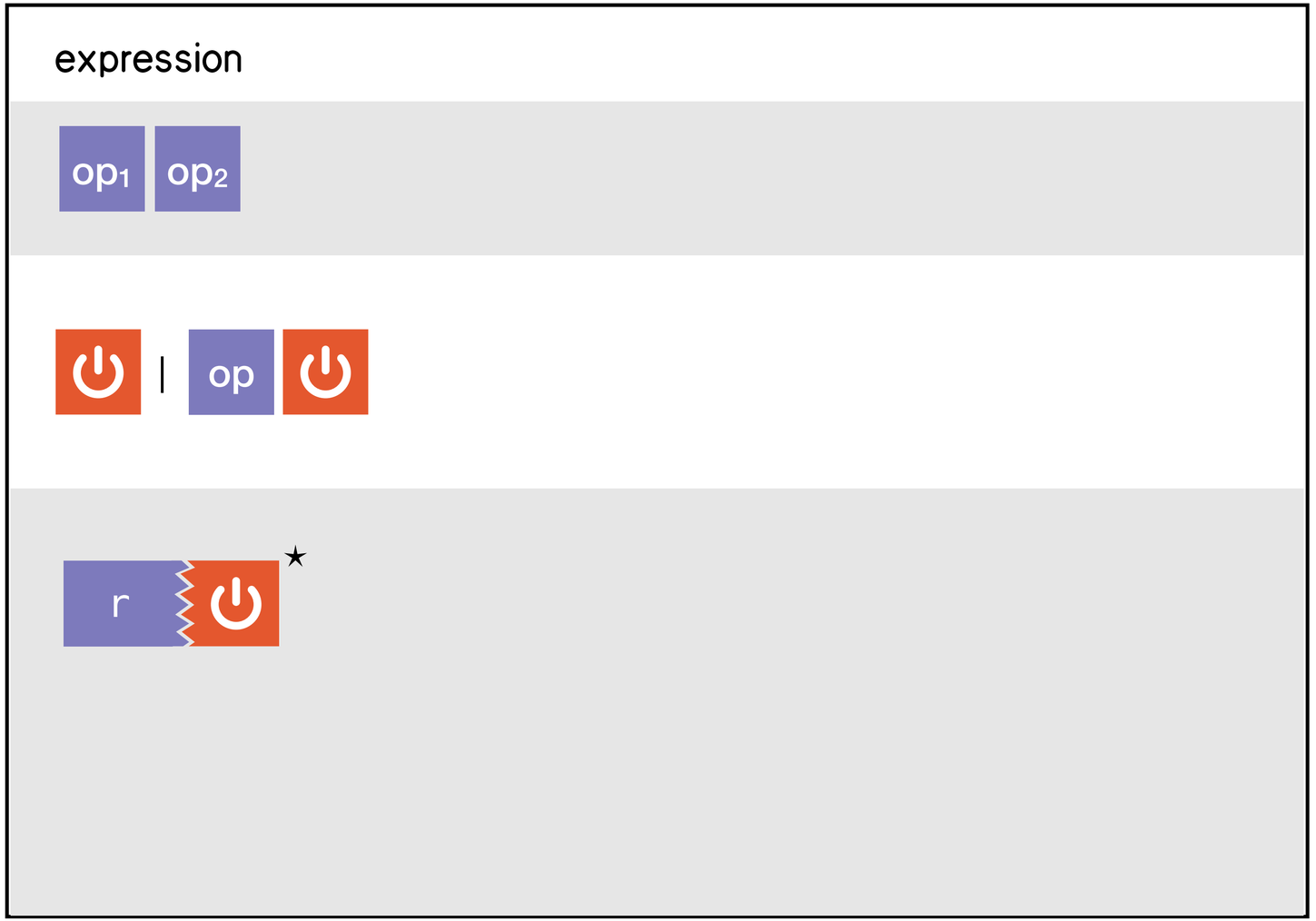
implies →

Trace inclusion

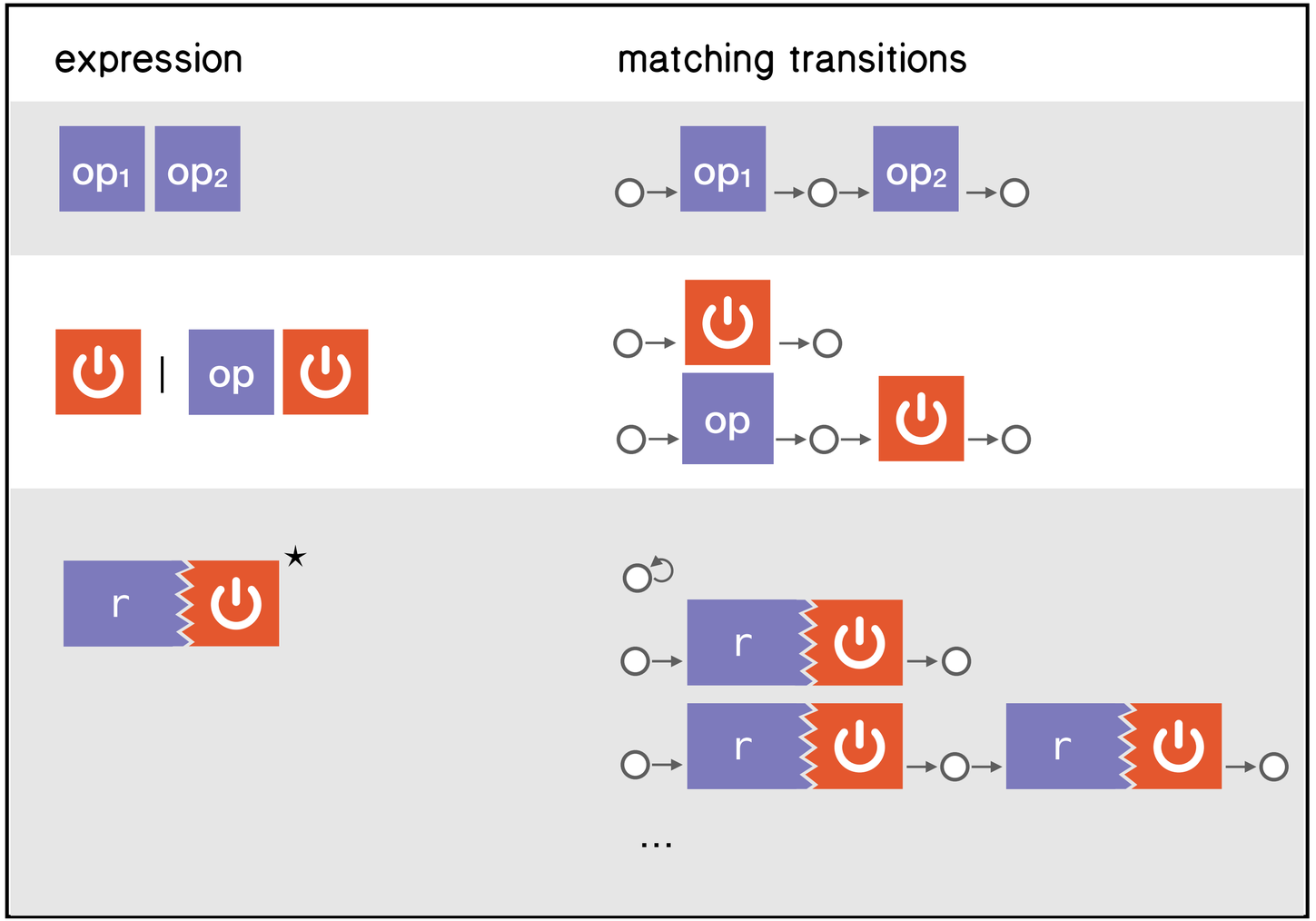
specification behavior
⊇
running code behavior

Composition theorem

Kleene algebra for transition relations

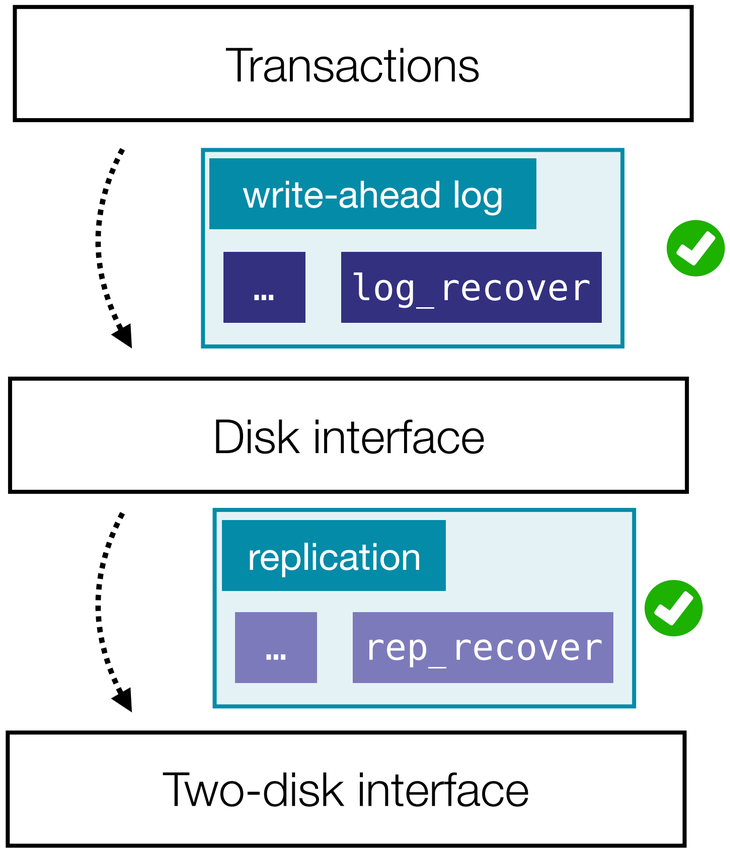


Kleene algebra for transition relations

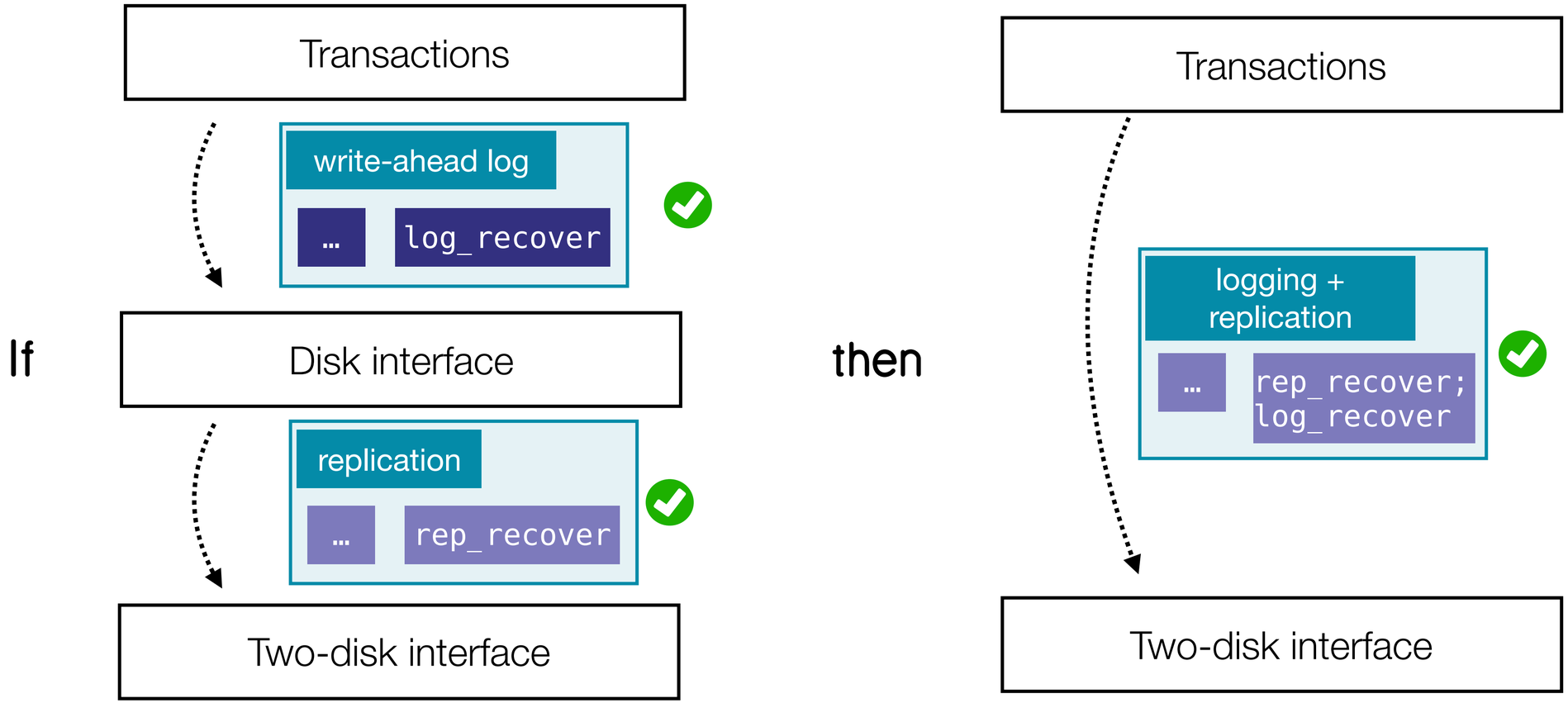


Theorem: recovery refinements compose

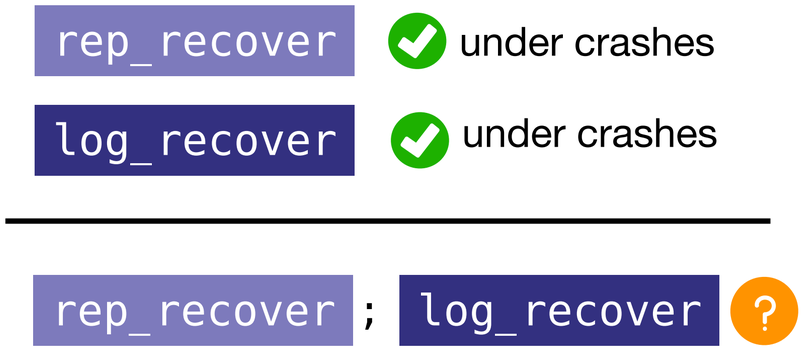
If



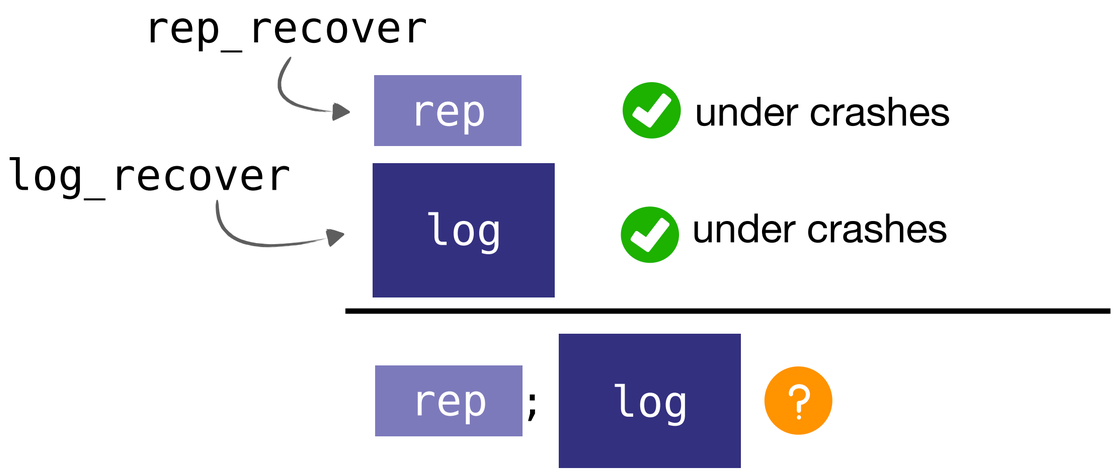
Theorem: recovery refinements compose

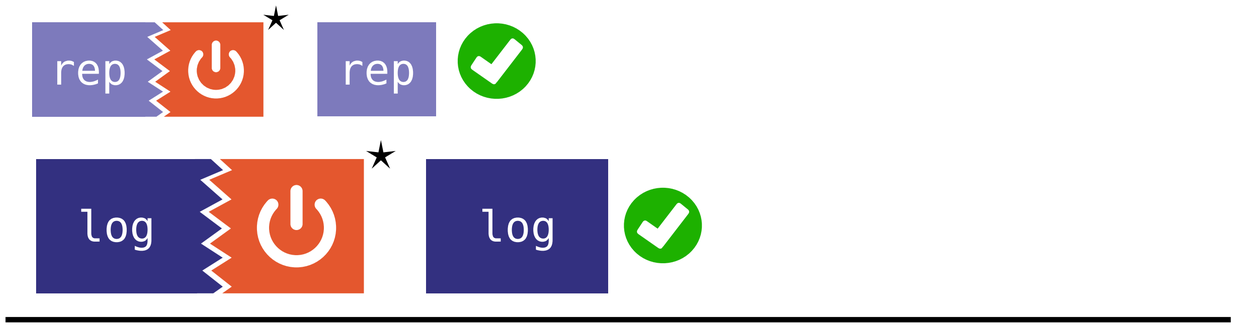


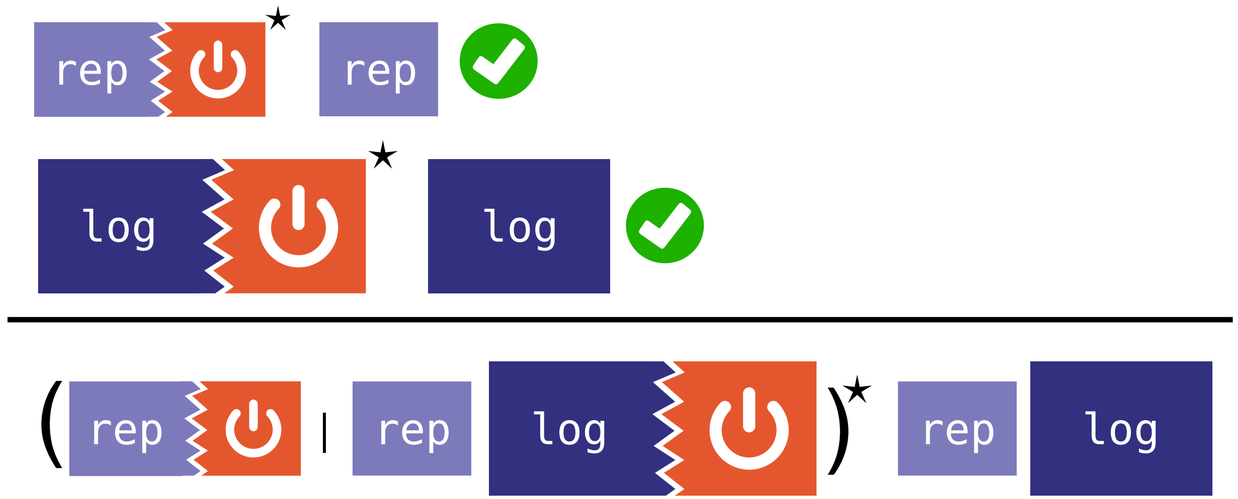
Goal: prove composed recovery correct

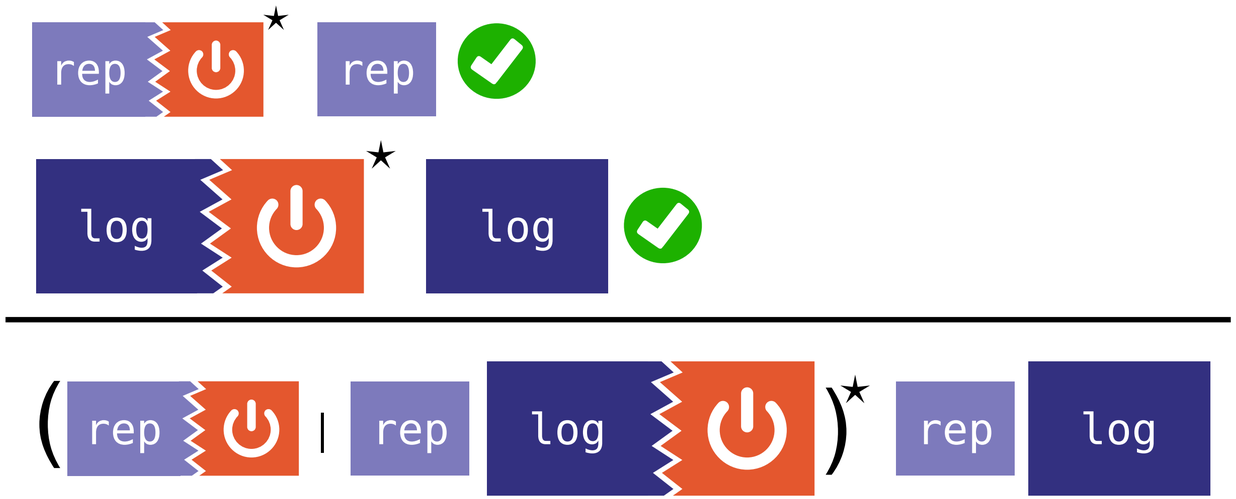


Goal: prove composed recovery correct




rep
rep
log
log


rep
rep
log
log
(
rep
|
rep
log
)*
rep
log



how to re-use recovery proofs here?

Using Kleene algebra for reasoning

Using Kleene algebra for reasoning

after *de-nesting* $(p \mid q)^\star = p^\star(qp^\star)^\star$

Using Kleene algebra for reasoning

after *de-nesting* $(p \mid q)^\star = p^\star(qp^\star)^\star$

Using Kleene algebra for reasoning

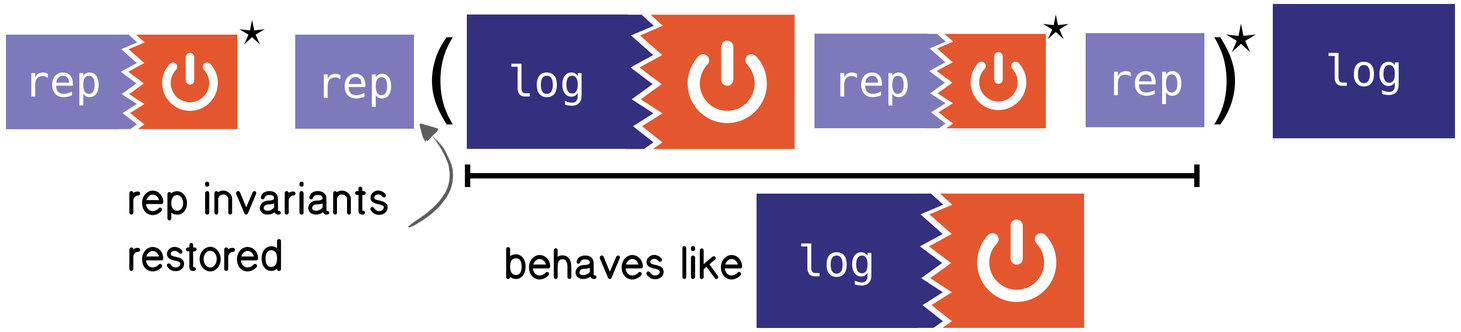
after *de-nesting* $(p \mid q)^\star = p^\star(qp^\star)^\star$

after *sliding* $(pq)^\star p = p(qp)^\star$

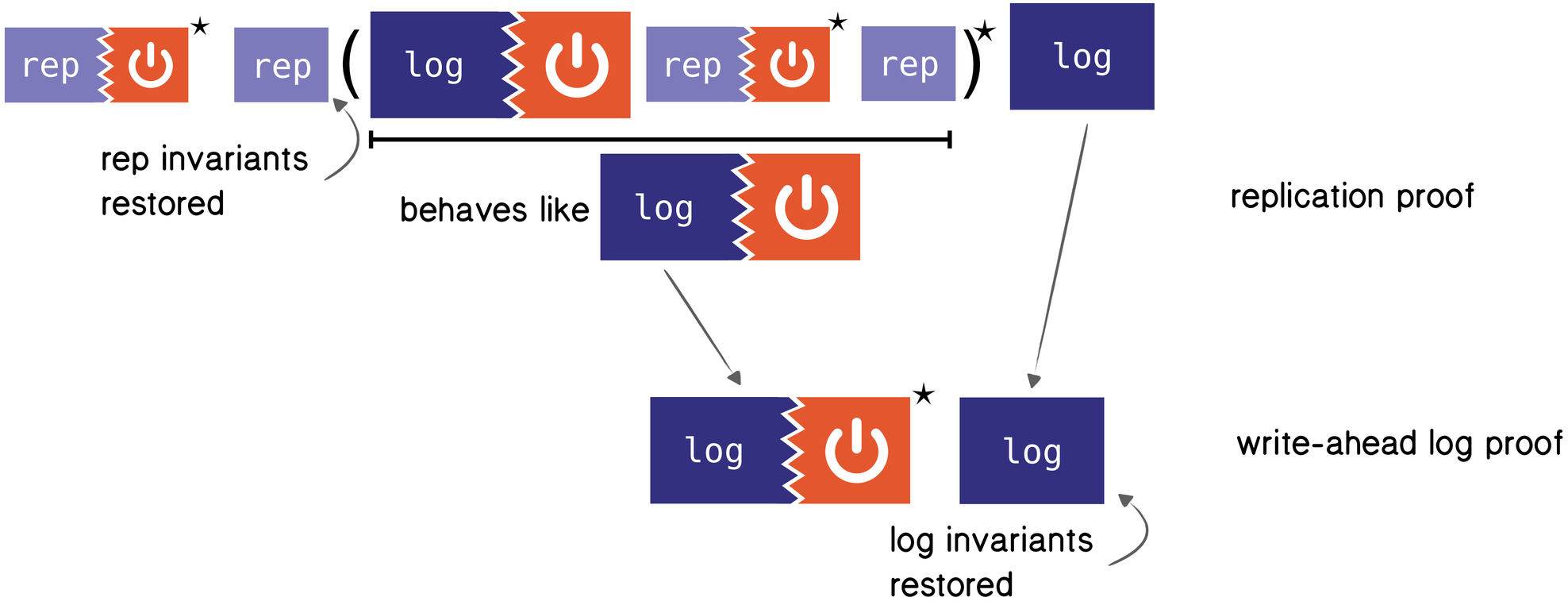
After rewrite both proofs apply

After rewrite both proofs apply

After rewrite both proofs apply



After rewrite both proofs apply



Argosy is implemented and verified in Coq

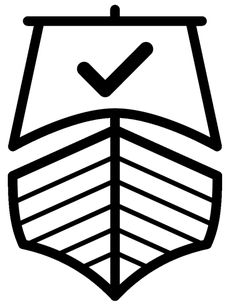
3,200 lines for framework

4,000 lines for verified example (logging + replication)

Example extracts to Haskell and runs

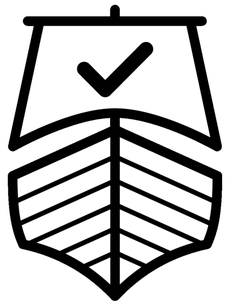
github.com/mit-pdos/argosy

Argosy: modular proofs of layered storage systems

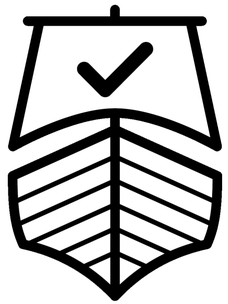


Argosy: modular proofs of layered storage systems

Kleene algebra

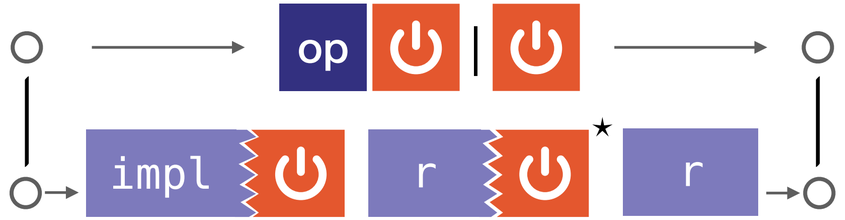


Argosy: modular proofs of layered storage systems

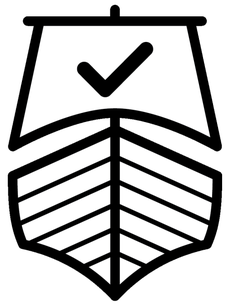


Kleene algebra

recovery refinement

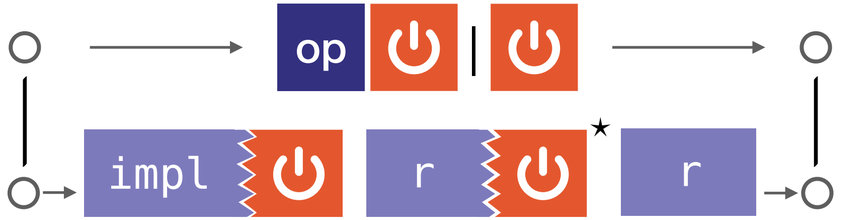


Argosy: modular proofs of layered storage systems

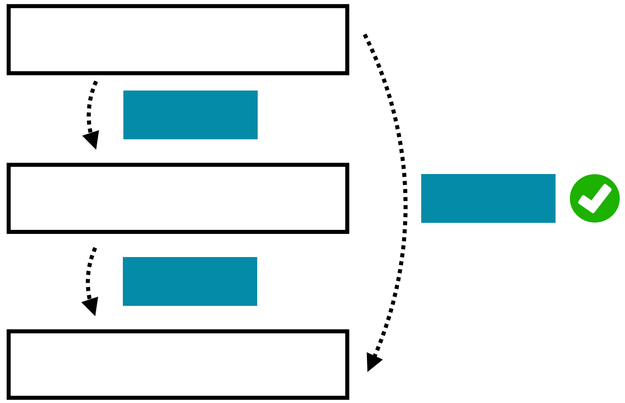


Kleene algebra

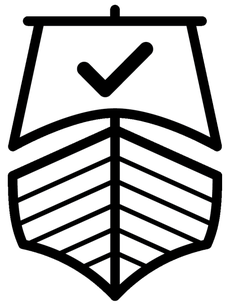
recovery refinement



modular proofs



Argosy: modular proofs of layered storage systems

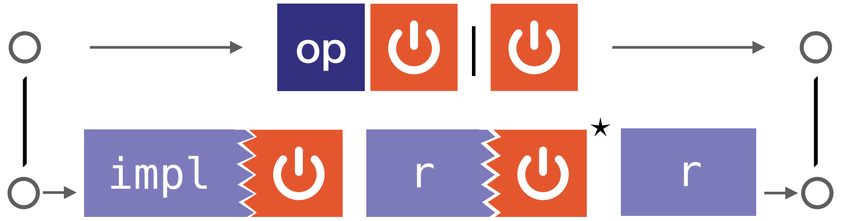


come find us after!

↳ Tej and Joe

Kleene algebra

recovery refinement



modular proofs