galois

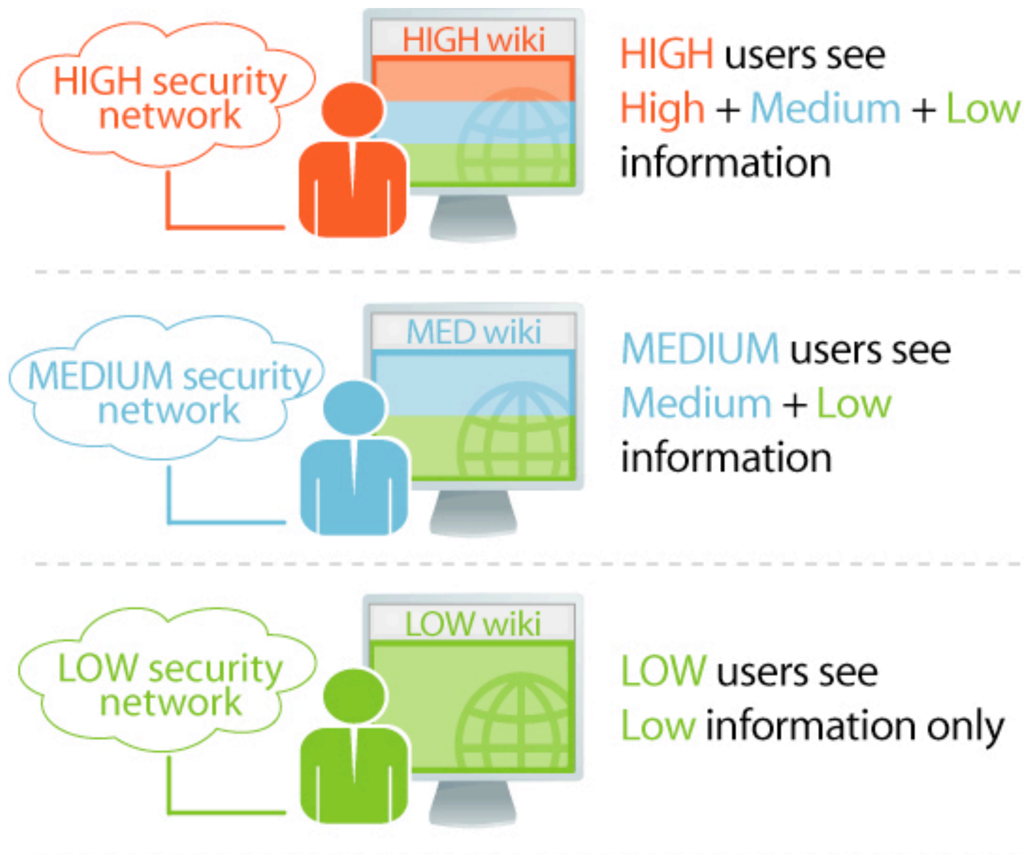
Analyzing a Cross-Domain Component: Lessons Learned and Future Directions

John Matthews

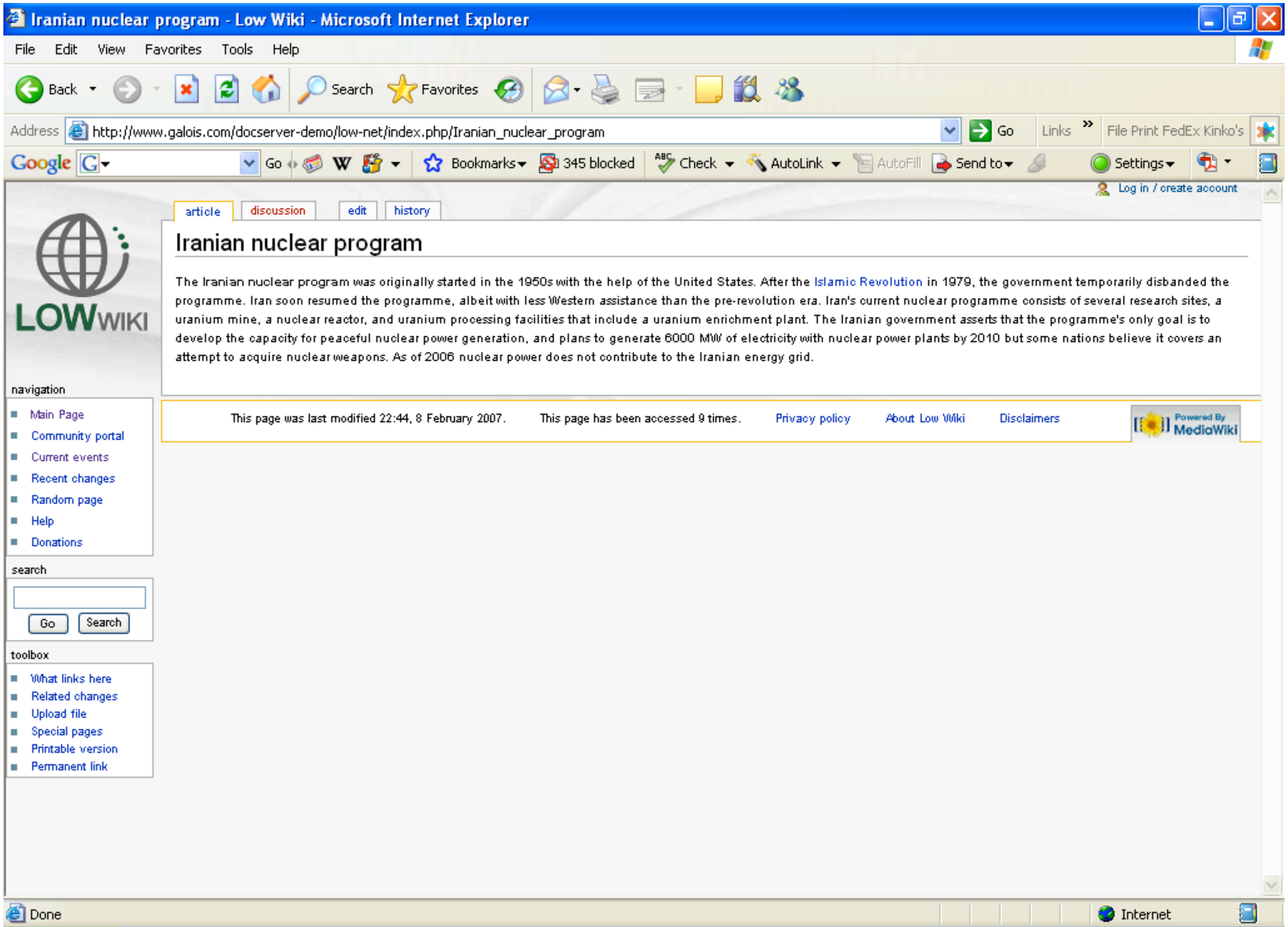
Joint work with Levent Erkök, Paul Graunke, Joe Hurd, Dylan McNamee, Lee Pike, Joel Stanley, Aaron Tomb

matthews@galois.com

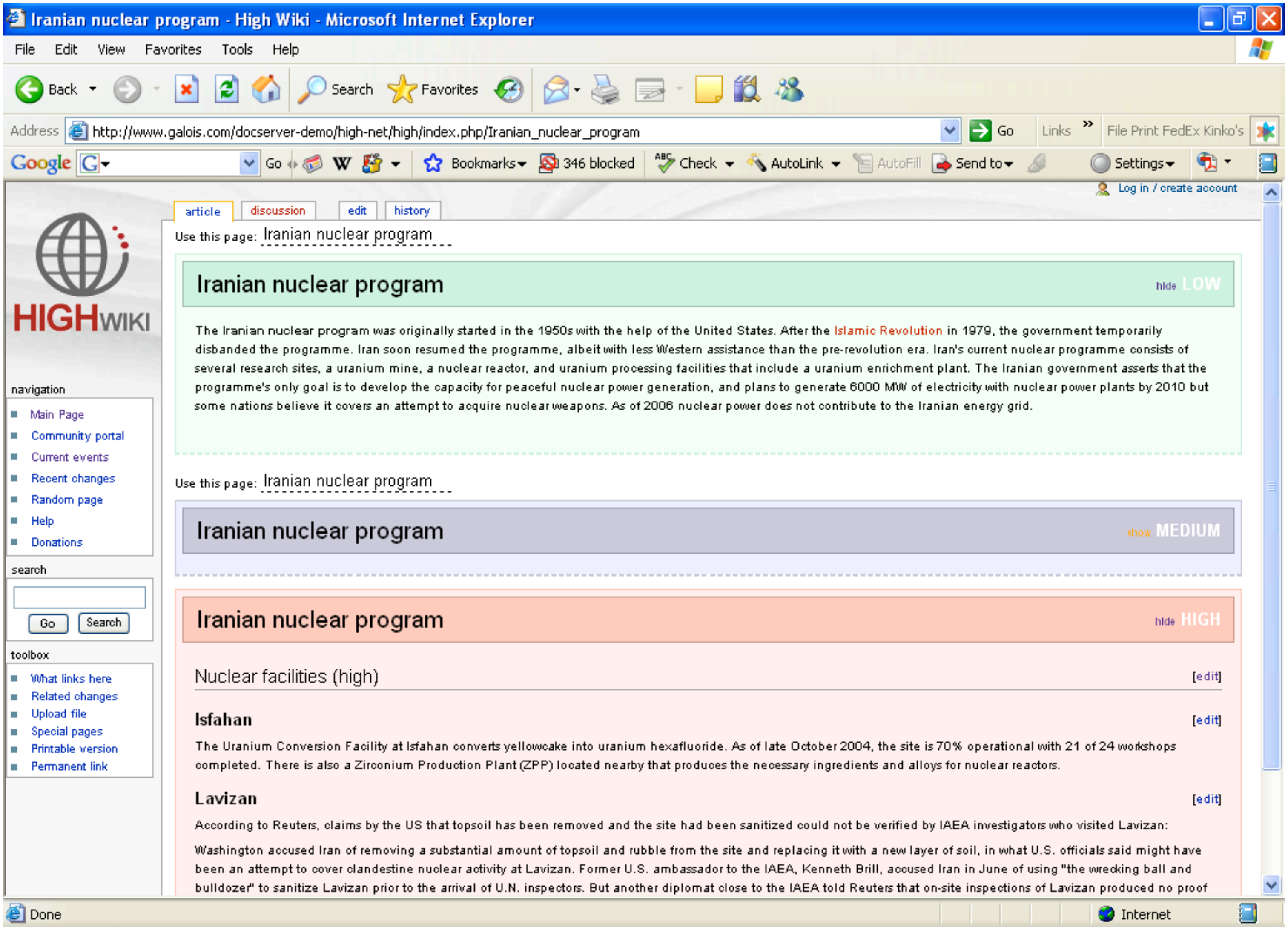
Tearline Wiki: Cross-domain collaboration service



Wikis: editable knowledge repositories



SBIR DATA RIGHTS || **Contract No.:** *N00039-05C-0036* || **Contractor Name**: *Galois Connections, Inc.* || **Contractor Address:** *12725 SW Millikan Way, Ste 290, Beaverton OR 97005* || *Expiration of SBIR Data Rights Period: 5 years following April 5, 2006 or final delivery of subsequent extensions to this project.* || *The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software-- Small Business Innovative Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.*



Iranian nuclear program - High Wiki - Microsoft Internet Explorer

http://www.galois.com/docserver-demo/high-net/high/index.php/Iranian_nuclear_program

article | discussion | edit | history

Use this page: Iranian nuclear program

Iranian nuclear program

hide LOW

The Iranian nuclear program was originally started in the 1950s with the help of the United States. After the Islamic Revolution in 1979, the government temporarily disbanded the programme. Iran soon resumed the programme, albeit with less Western assistance than the pre-revolution era. Iran's current nuclear programme consists of several research sites, a uranium mine, a nuclear reactor, and uranium processing facilities that include a uranium enrichment plant. The Iranian government asserts that the programme's only goal is to develop the capacity for peaceful nuclear power generation, and plans to generate 6000 MW of electricity with nuclear power plants by 2010 but some nations believe it covers an attempt to acquire nuclear weapons. As of 2006 nuclear power does not contribute to the Iranian energy grid.

Use this page: Iranian nuclear program

Iranian nuclear program

show MEDIUM

Iranian nuclear program

hide HIGH

Nuclear facilities (high)

[edit]

Isfahan [edit]

The Uranium Conversion Facility at Isfahan converts yellowcake into uranium hexafluoride. As of late October 2004, the site is 70% operational with 21 of 24 workshops completed. There is also a Zirconium Production Plant (ZPP) located nearby that produces the necessary ingredients and alloys for nuclear reactors.

Lavizan [edit]

According to Reuters, claims by the US that topsoil has been removed and the site had been sanitized could not be verified by IAEA investigators who visited Lavizan:

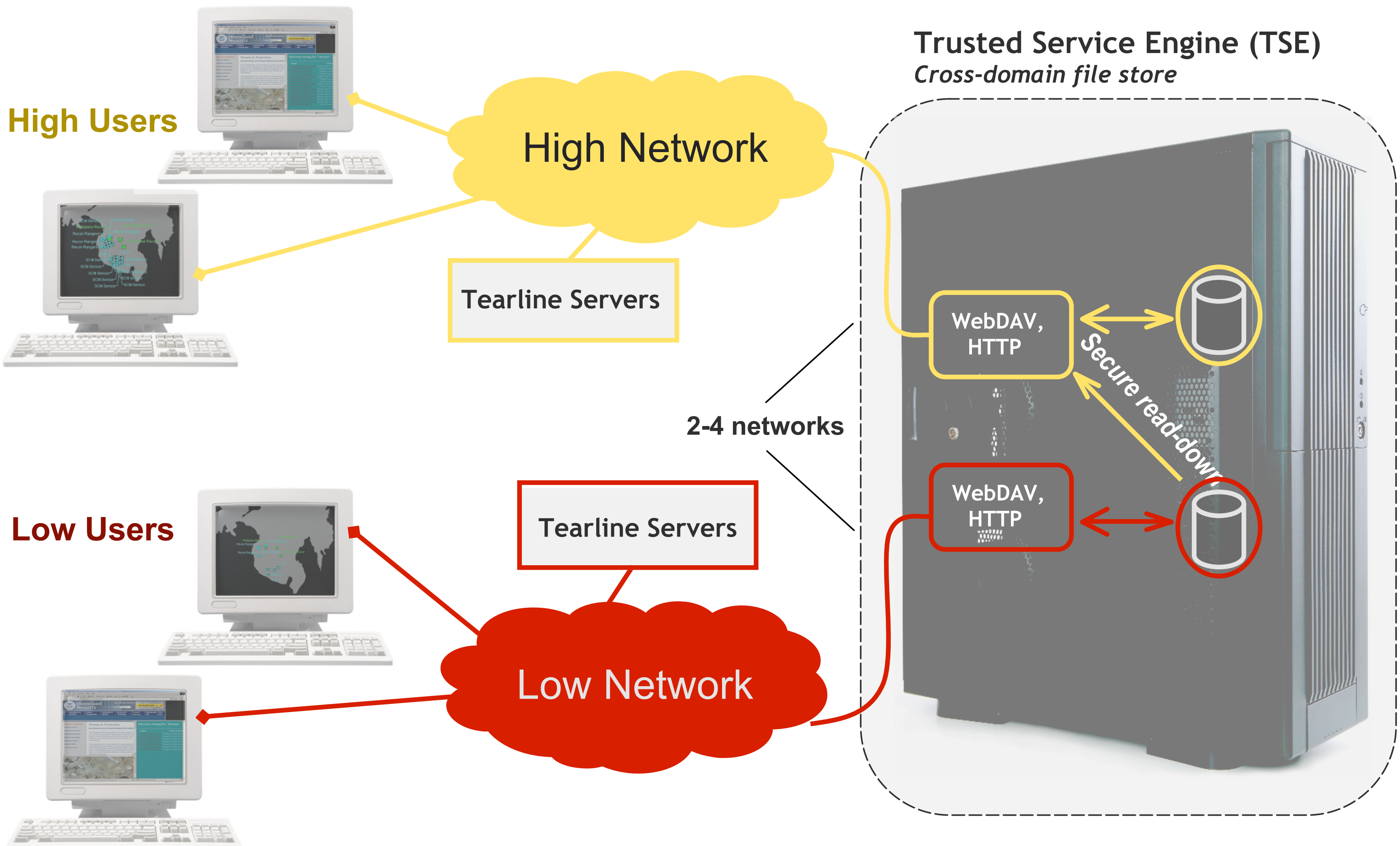
Washington accused Iran of removing a substantial amount of topsoil and rubble from the site and replacing it with a new layer of soil, in what U.S. officials said might have been an attempt to cover clandestine nuclear activity at Lavizan. Former U.S. ambassador to the IAEA, Kenneth Brill, accused Iran in June of using "the wrecking ball and bulldozer" to sanitize Lavizan prior to the arrival of U.N. inspectors. But another diplomat close to the IAEA told Reuters that on-site inspections of Lavizan produced no proof

SBIR DATA RIGHTS || **Contract No.:** *N00039-05C-0036* || **Contractor Name**: *Galois Connections, Inc.* || **Contractor Address:** *12725 SW Millikan Way, Ste 290, Beaverton OR 97005* || *Expiration of SBIR Data Rights Period: 5 years following April 5, 2006 or final delivery of subsequent extensions to this project.* || *The Government's rights to use, modify, reproduce, release, perform, display, or disclose technical data or computer software marked with this legend are restricted during the period shown as provided in paragraph (b)(4) of the Rights in Noncommercial Technical Data and Computer Software-- Small Business Innovative Research (SBIR) Program clause contained in the above identified contract. No restrictions apply after the expiration date shown above. Any reproduction of technical data, computer software, or portions thereof marked with this legend must also reproduce the markings.*

Outline

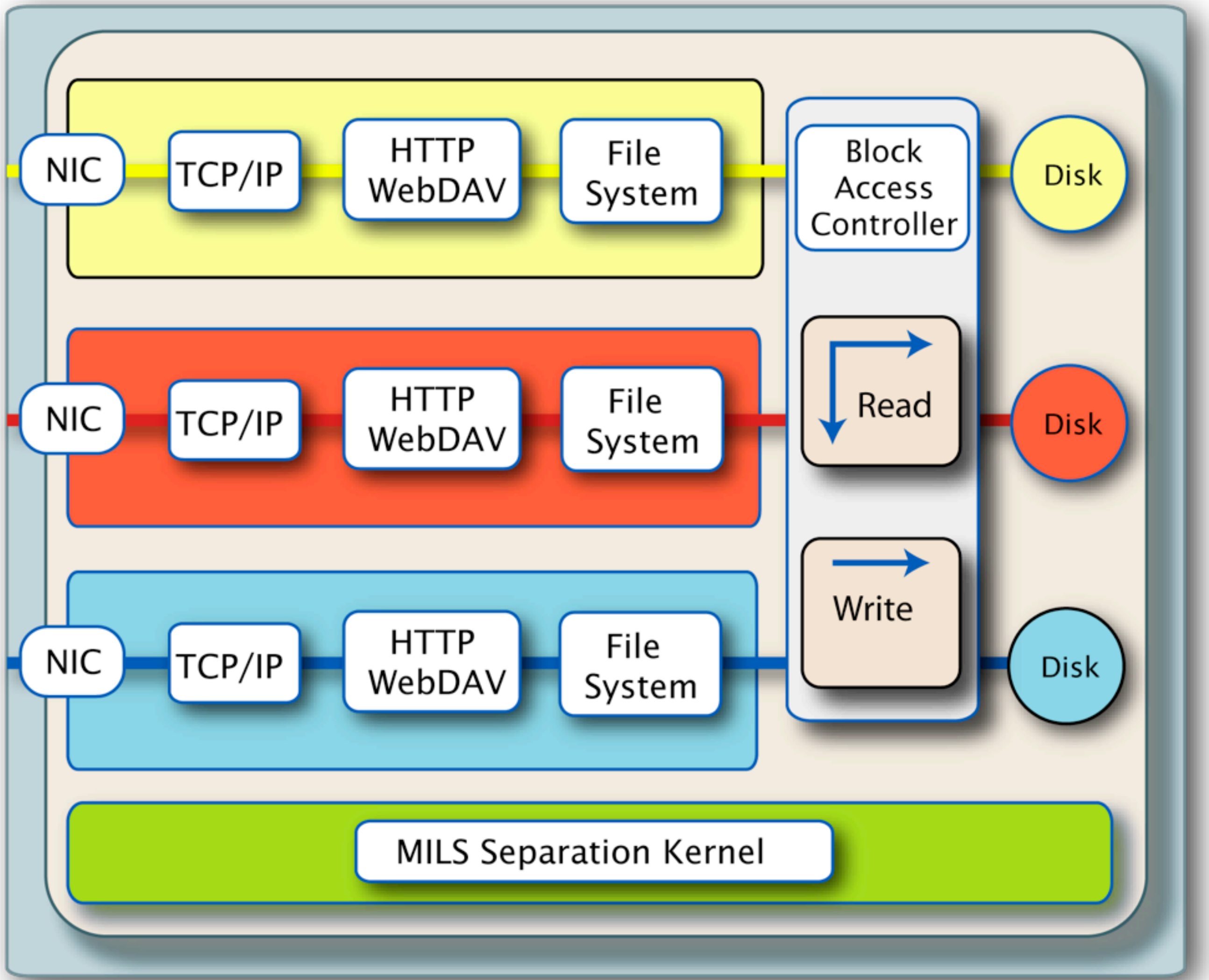
- ***Tearline Wiki* system architecture**
- Formally verifying the *Block Access Controller*
- Making future verifications easier

Tearline Wiki architecture

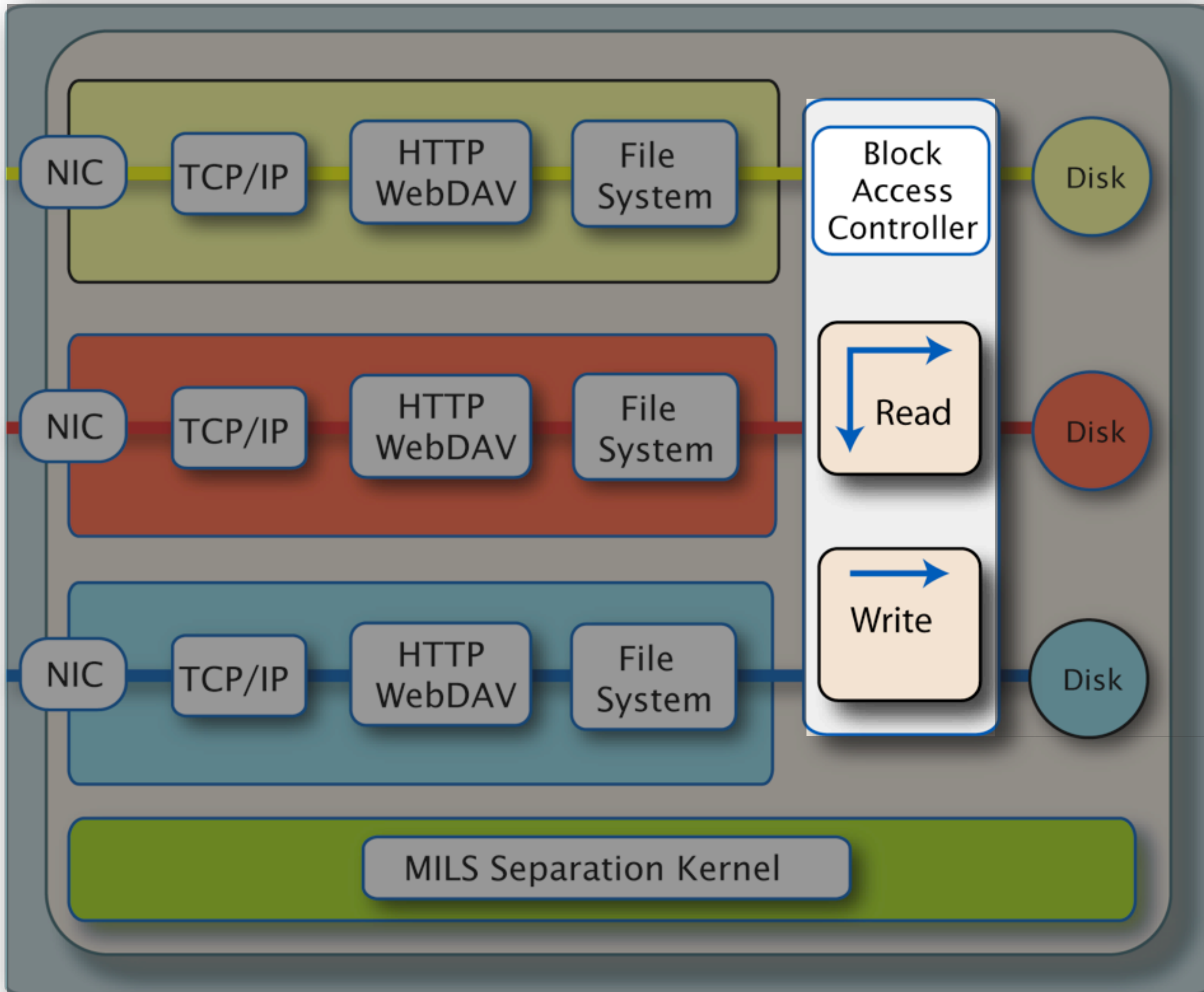


galois

TSE architecture



TSE architecture

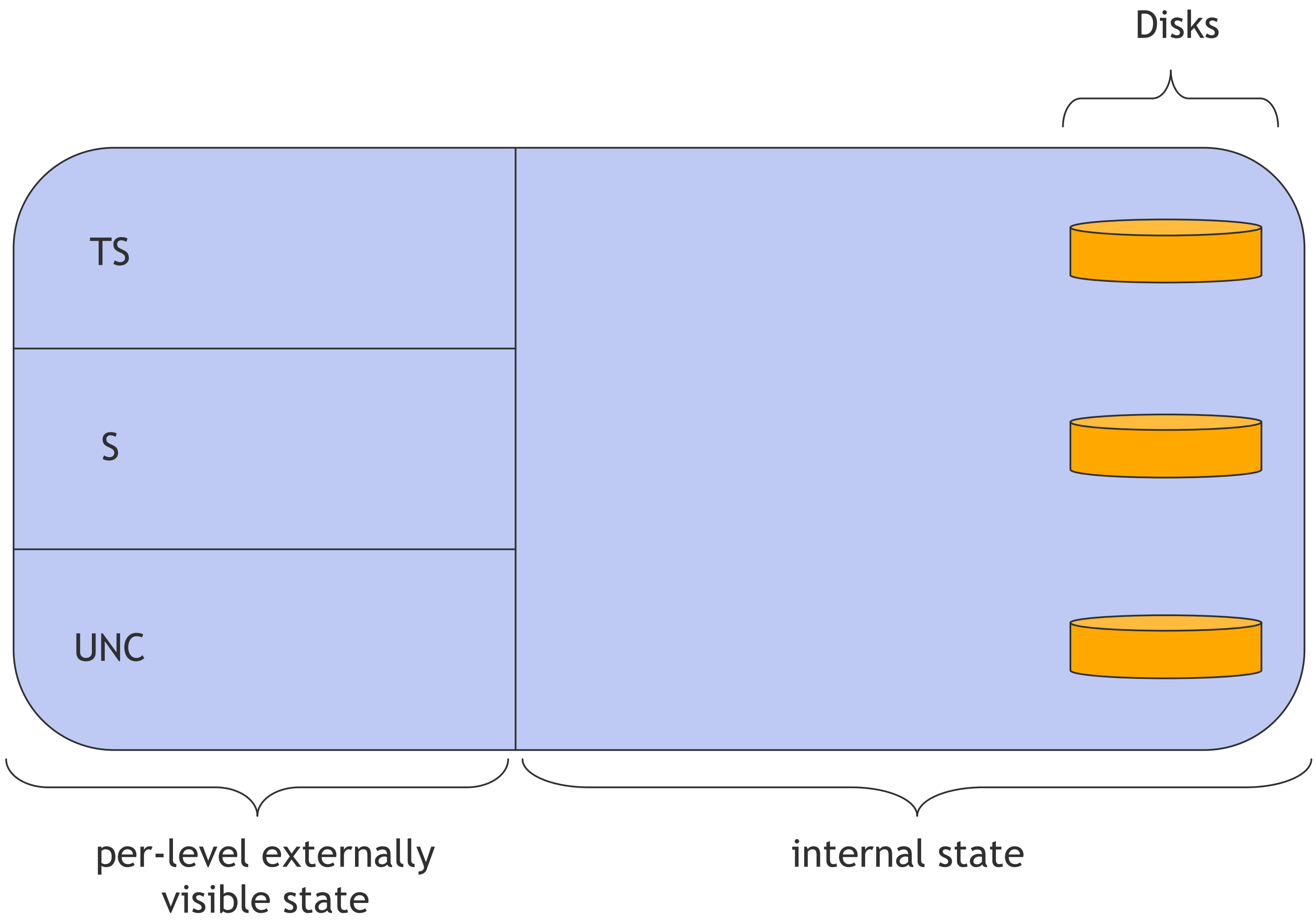


Block Access Controller (BAC)

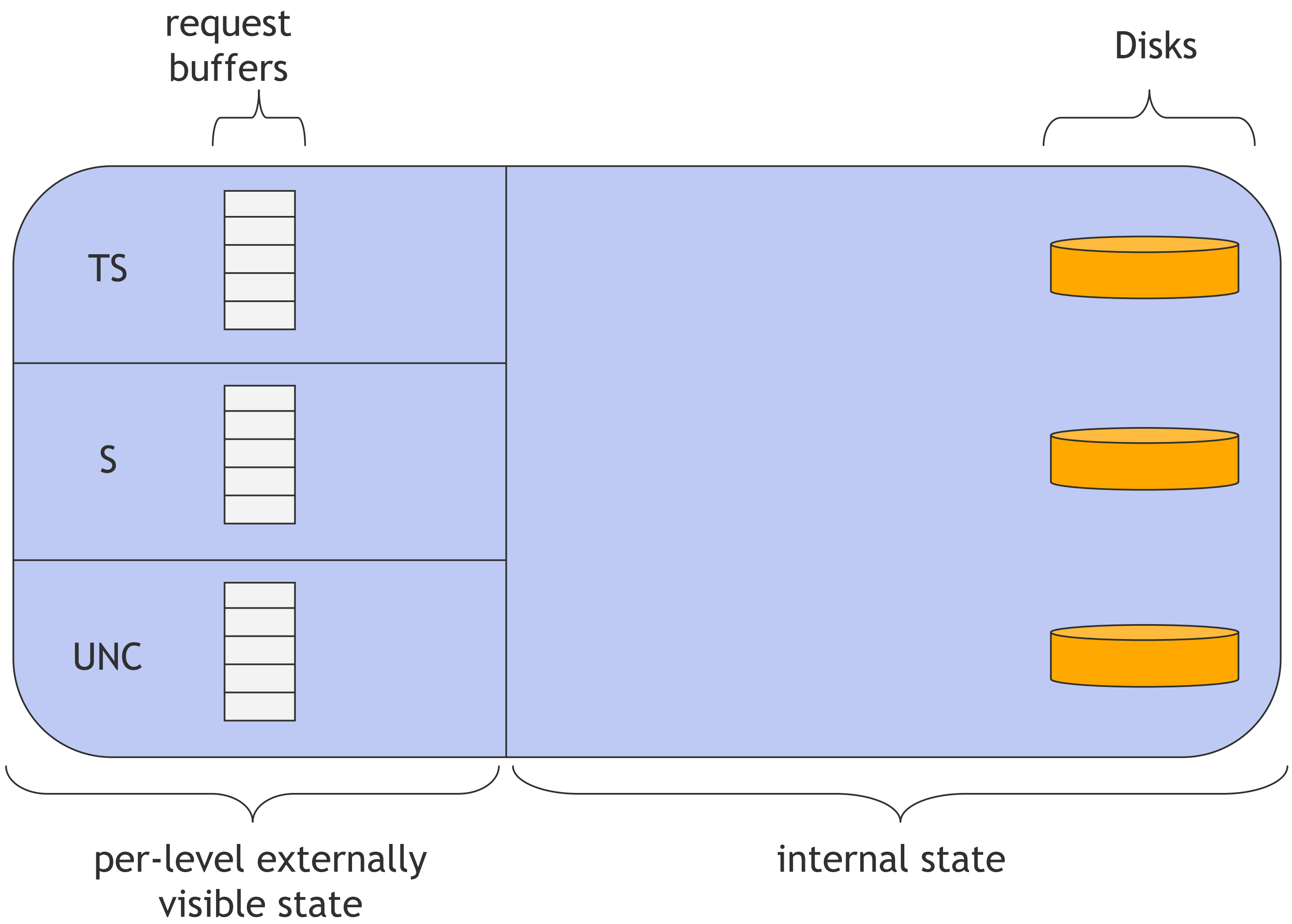
- BAC's functions
  - Mediate all disk block accesses
  - Connect single-level disks and partitions
  - Enforce Bell-LaPadula confidentiality rules
    - Reads from same or lower levels
    - Writes to same level (write-up not needed)

- Approximately 800 lines of generated C code

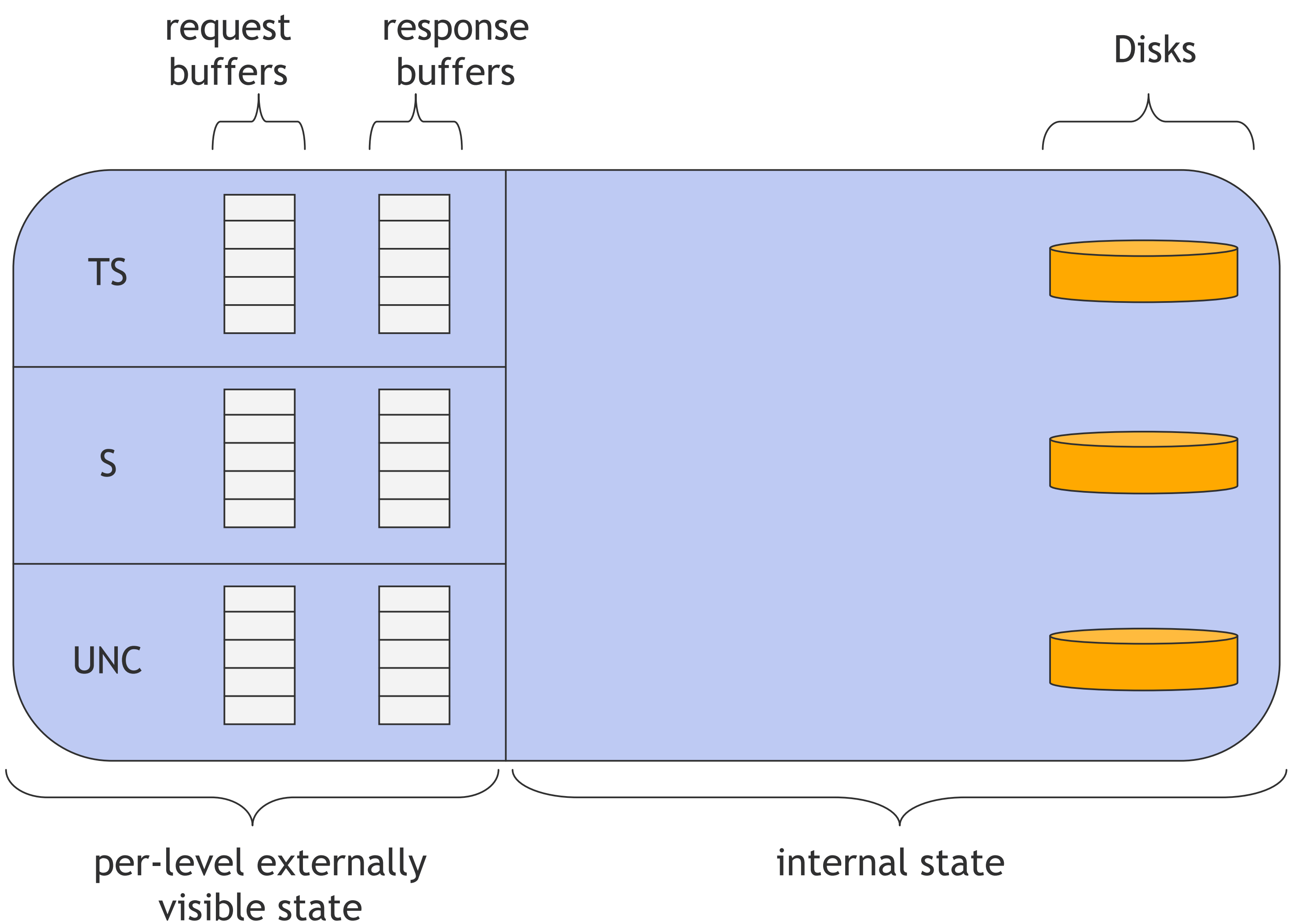
BAC state


Disks
TS
S
UNC
per-level externally visible state
internal state

BAC state

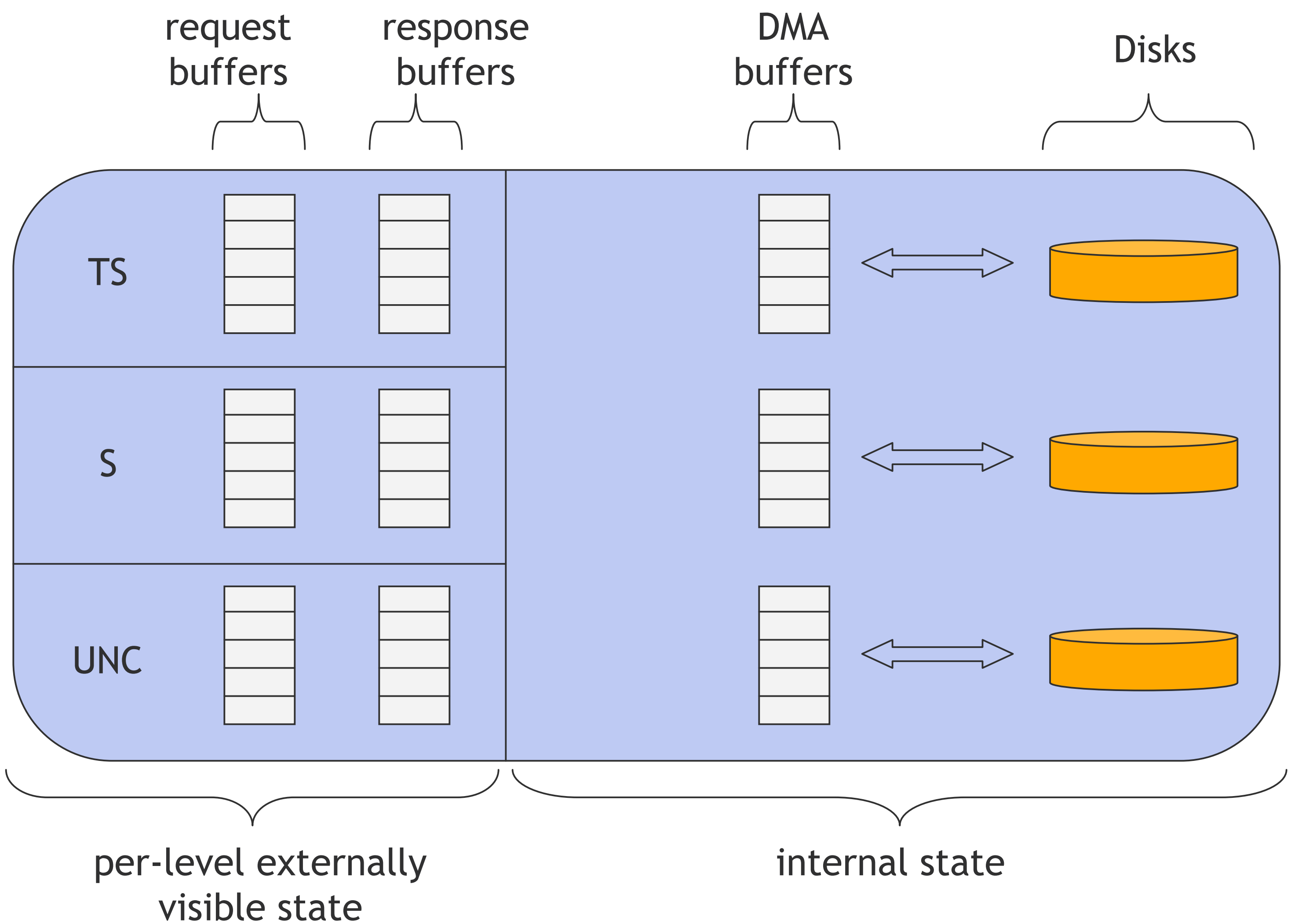

request buffers
Disks
TS
S
UNC
per-level externally visible state
internal state

BAC state



galois

BAC state



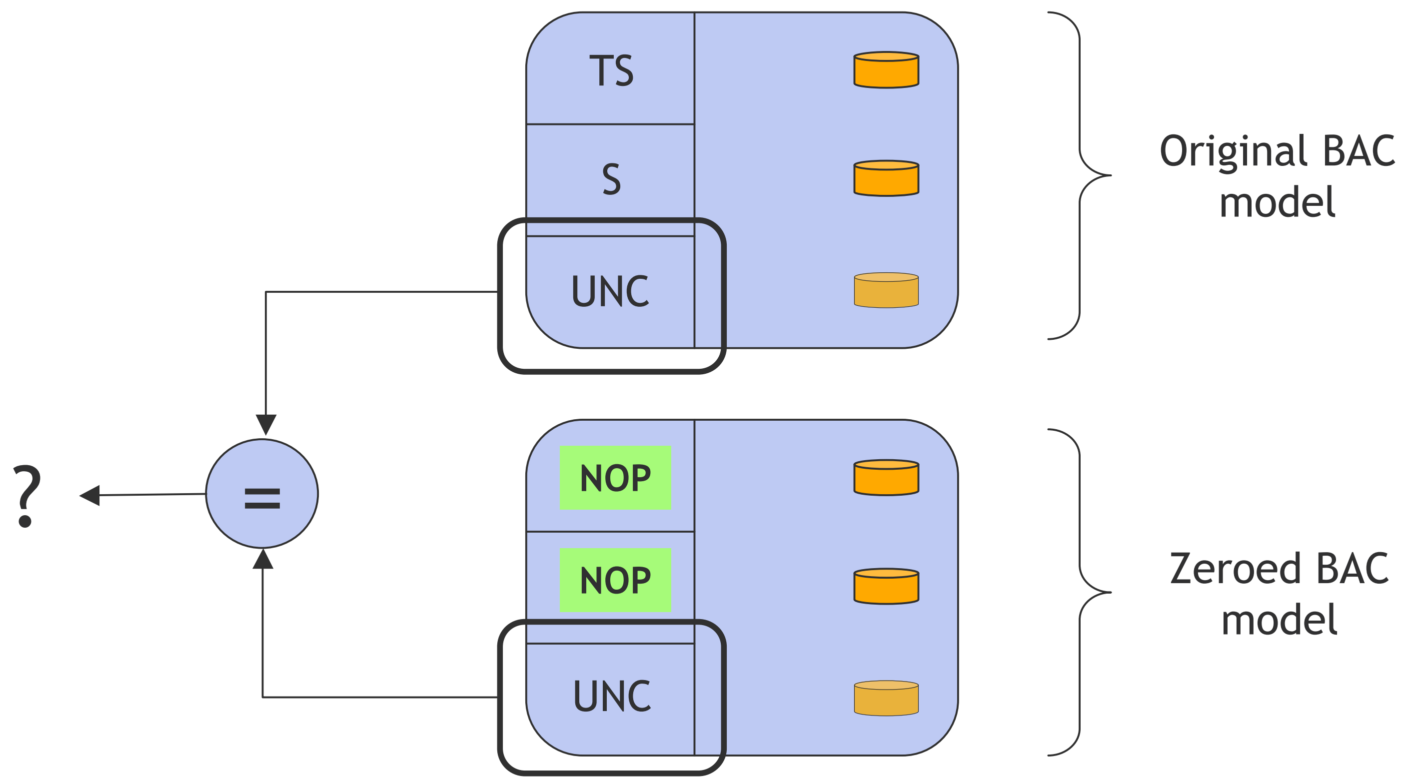
galois

Outline

- *Tearline Wiki* system architecture
- **Formally verifying the *Block Access Controller***
- Making future verifications easier

BAC verification approach

- We want EAL7-strength assurance evidence, so we formally verified:
  - *Safety*: BAC never transitions to an error state
  - *Data separation*: BAC's output buffer values are not dependent on any higher-security input buffer values



galois

BAC verification approach

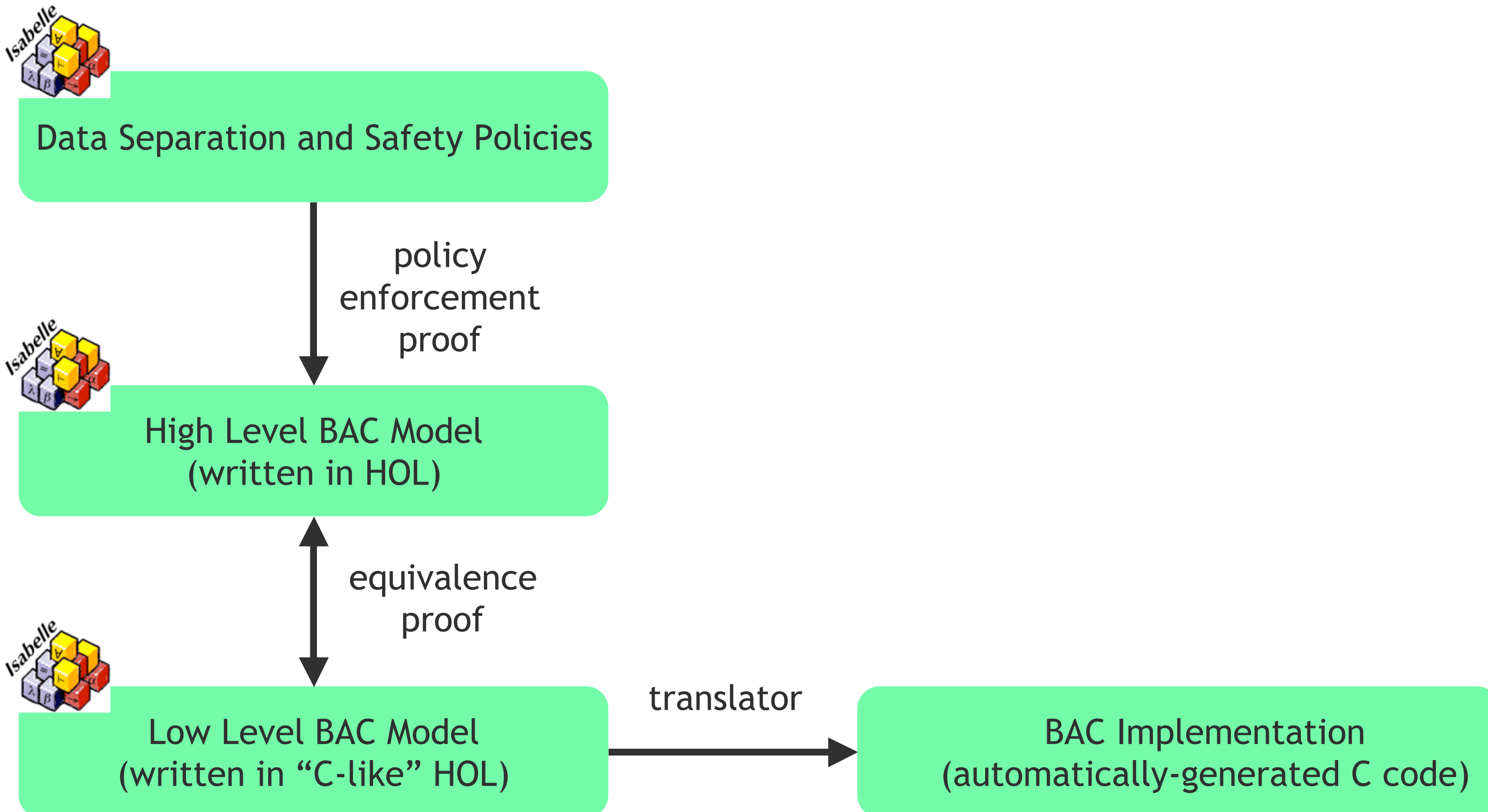
- Originally we tried to formally verify these properties with model checkers
  - But they timed out due to state space explosion

- So we switched to using Isabelle theorem prover
  - Feasible, since BAC implementation is only 800 lines long

- Isabelle is attractive for EAL7 assurance evidence
    - Small proof kernel
    - Proof kernel can generate independently-checkable proof objects
    - Records all axioms a theorem depends on

- Data separation proof inspired by [von Oheimb, ESORICS'04]

BAC assurance evidence

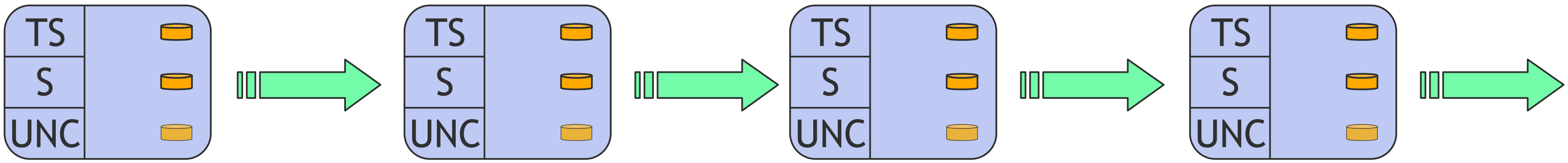


BAC runtime safety

- To prove data separation, we first had to prove no error states are reachable
  - Out-of-bounds array access
  - Out-of bounds disk block ID
  - Access to memory undergoing DMA transfer
  - Too many simultaneous DMA transfers to a single disk
  - Multiple simultaneous DMA transfers to same memory region

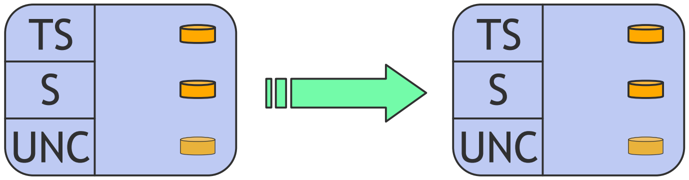
- Each possible error state had to be turned into a *loop invariant*: a property that
  - Is true of the BAC's initial state
  - Remains true each time around the top-level BAC event loop

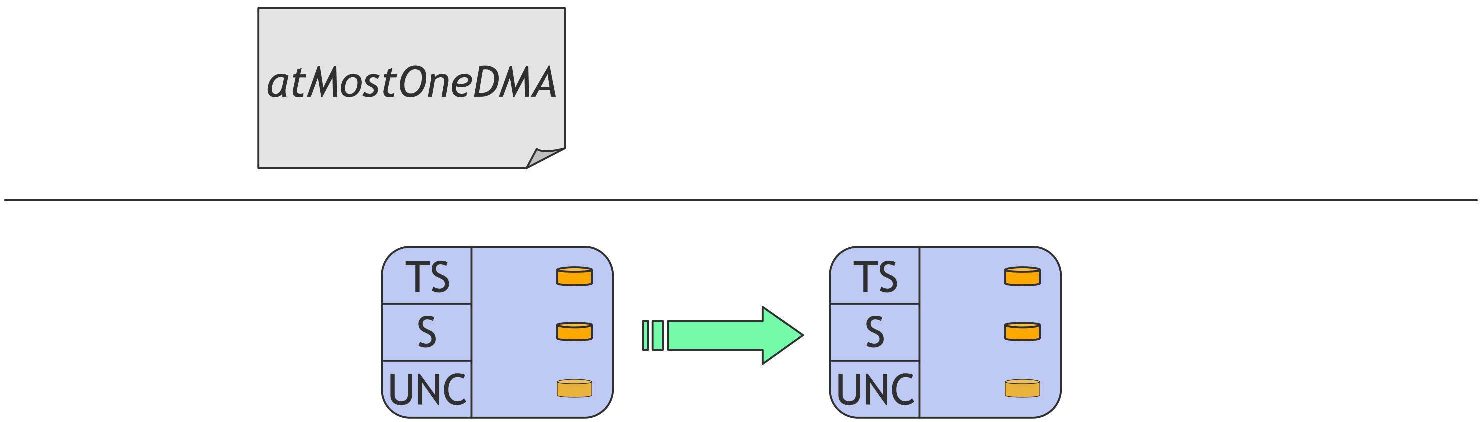
- Example
  - *atMostOneDMA*: "There is at most one DMA transfer occurring to any given memory page"

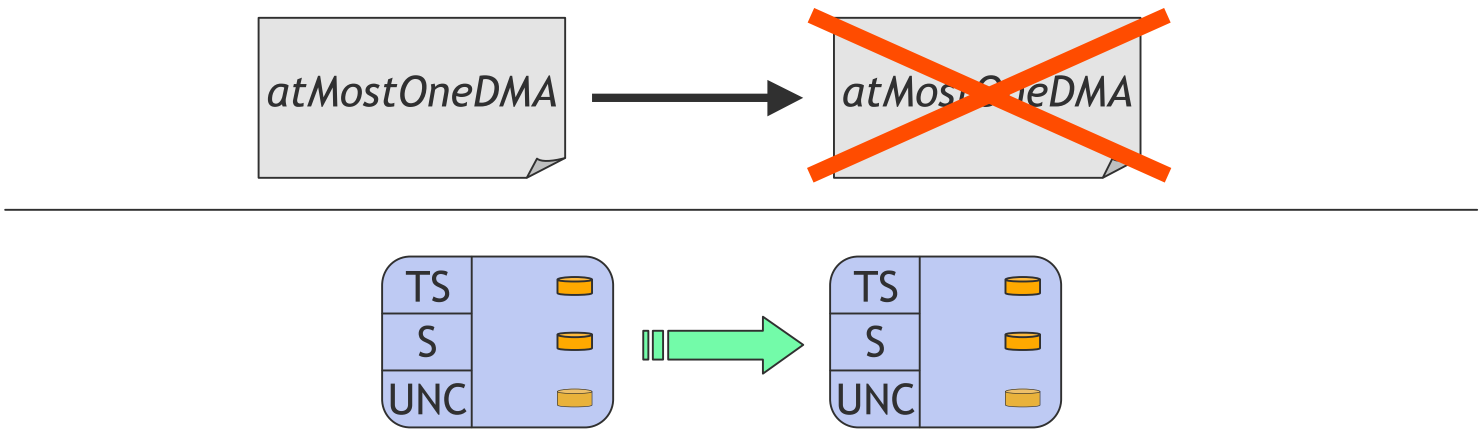


A key challenge in BAC proofs

- Finding appropriate loop invariants took too long
- Invariants are often correct, but not *inductive*
  - Need to perform unknown number of manual *invariant strengthening* steps, until inductive invariant is found

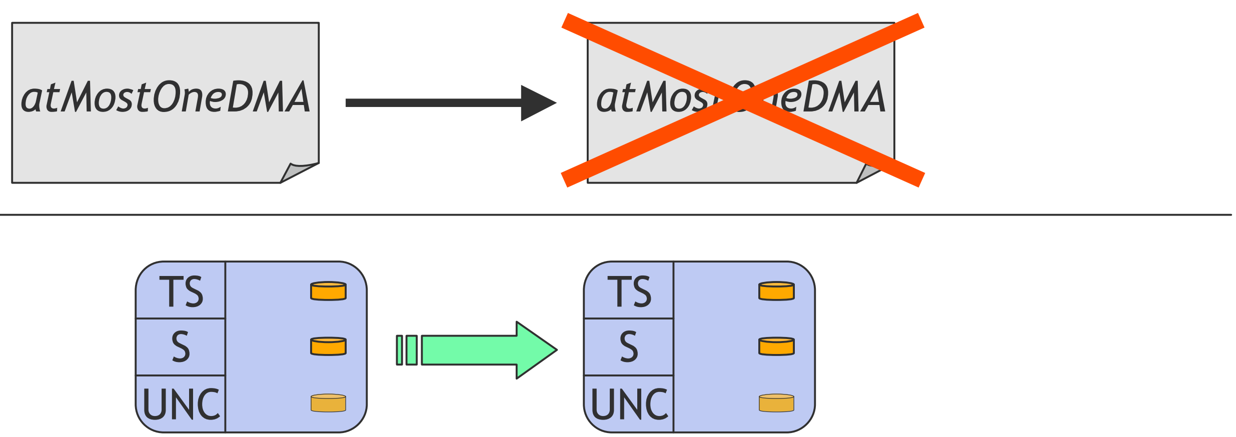

TS
S
UNC
TS
S
UNC


atMostOneDMA
TS
S
UNC
TS
S
UNC



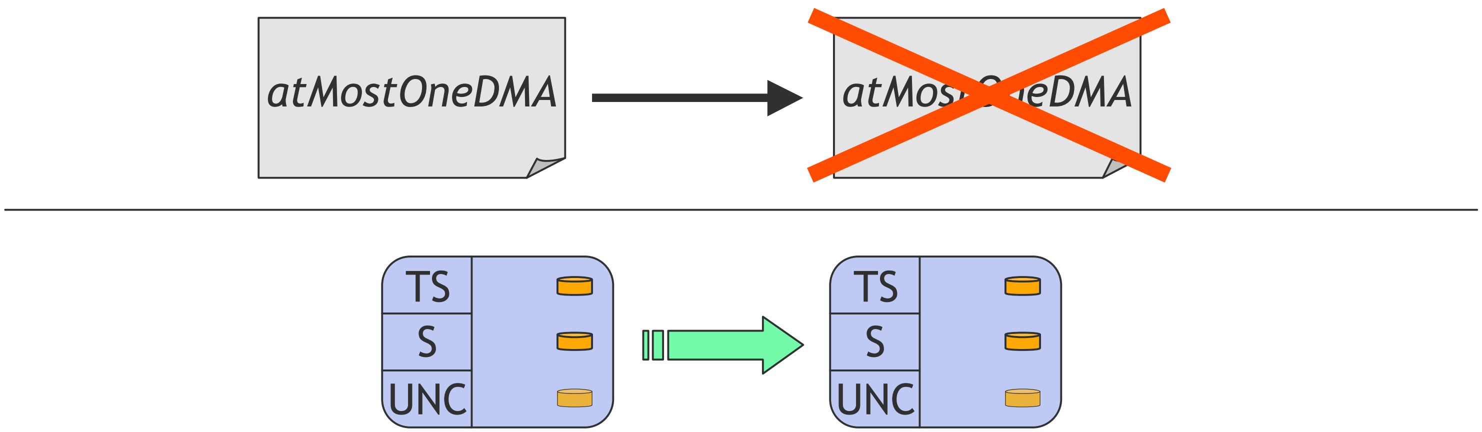
galois

- When induction step proof fails, there are two possibilities:
  - Case 1: before-state is reachable --> **invariant is too strong (i.e. false)**



galois

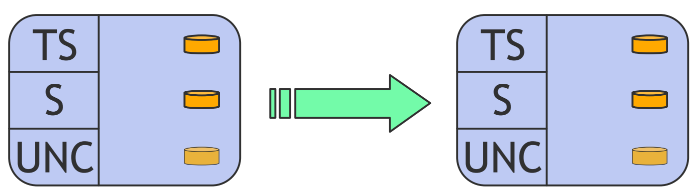
- When induction step proof fails, there are two possibilities:
  - Case 1: before-state is reachable --> invariant is too strong (i.e. false)
  - Case 2: before-state is unreachable --> **invariant is too weak**

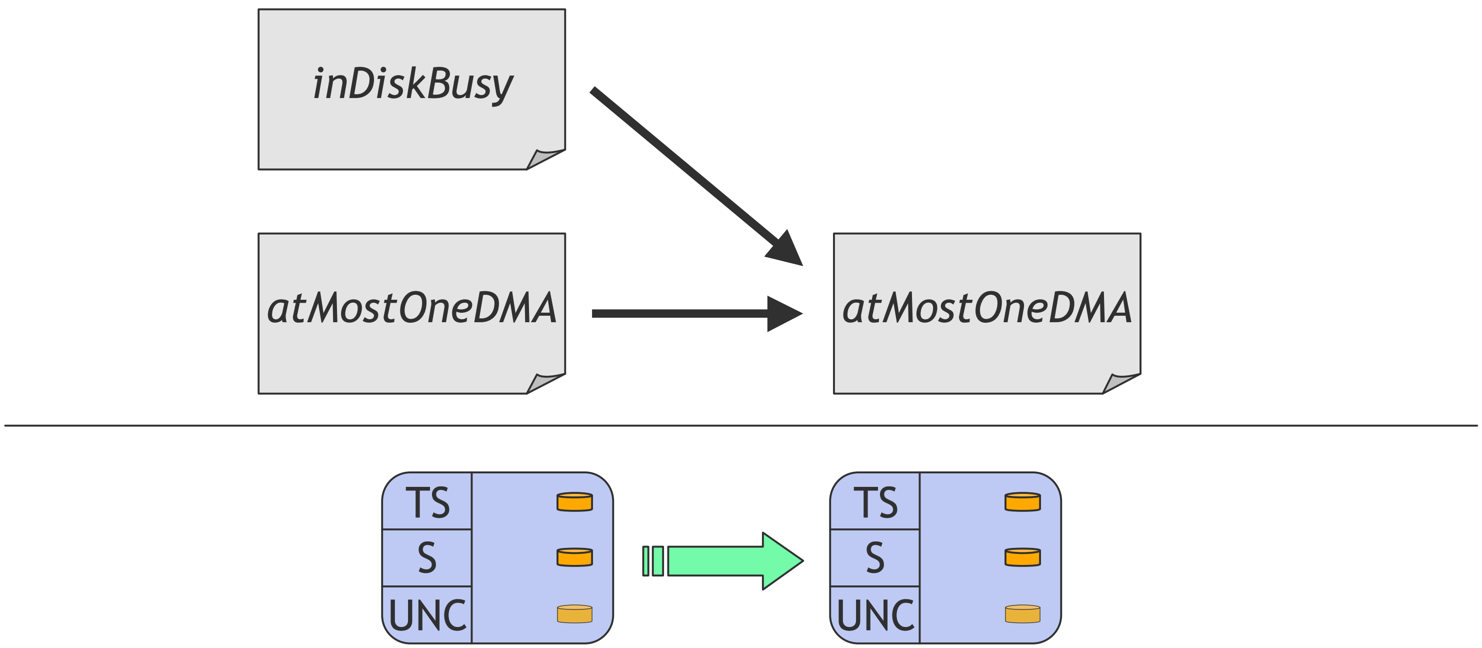


| galois |

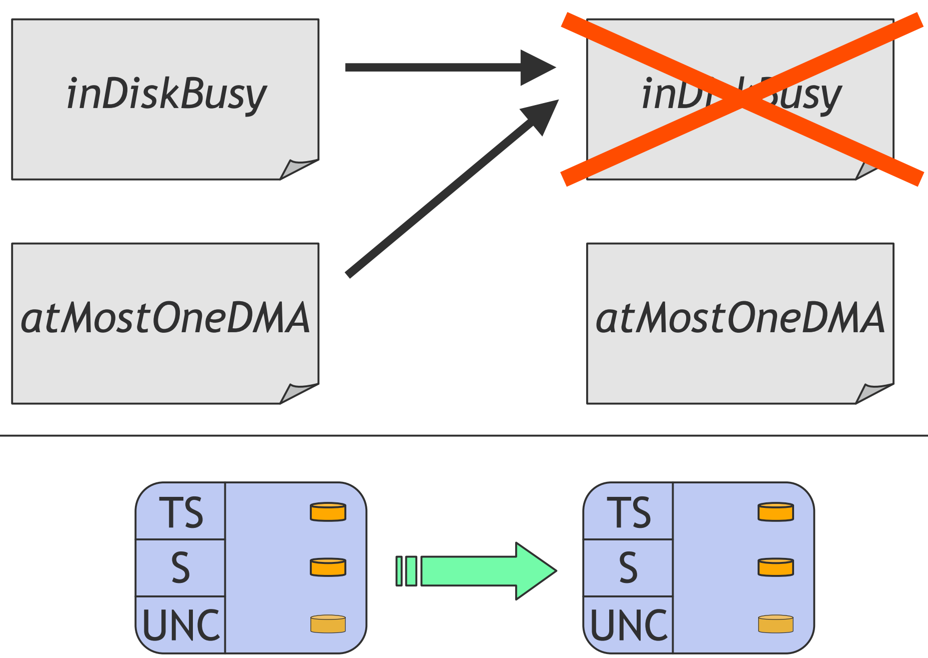
inDiskBusy

atMostOneDMA



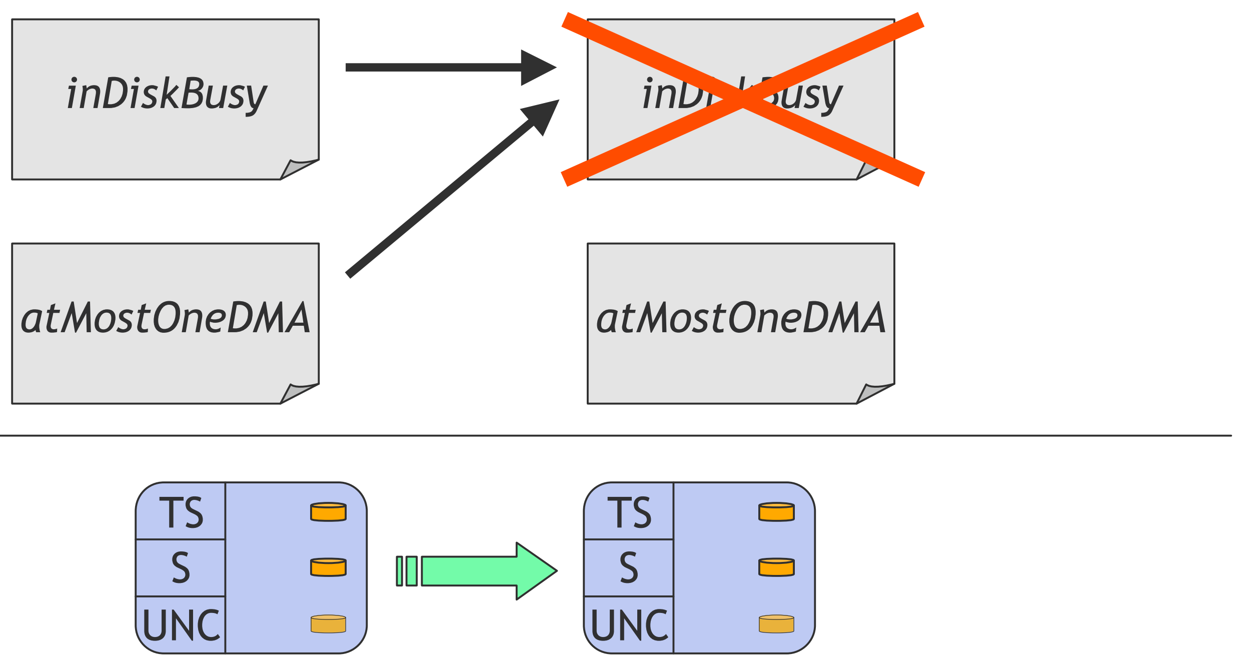


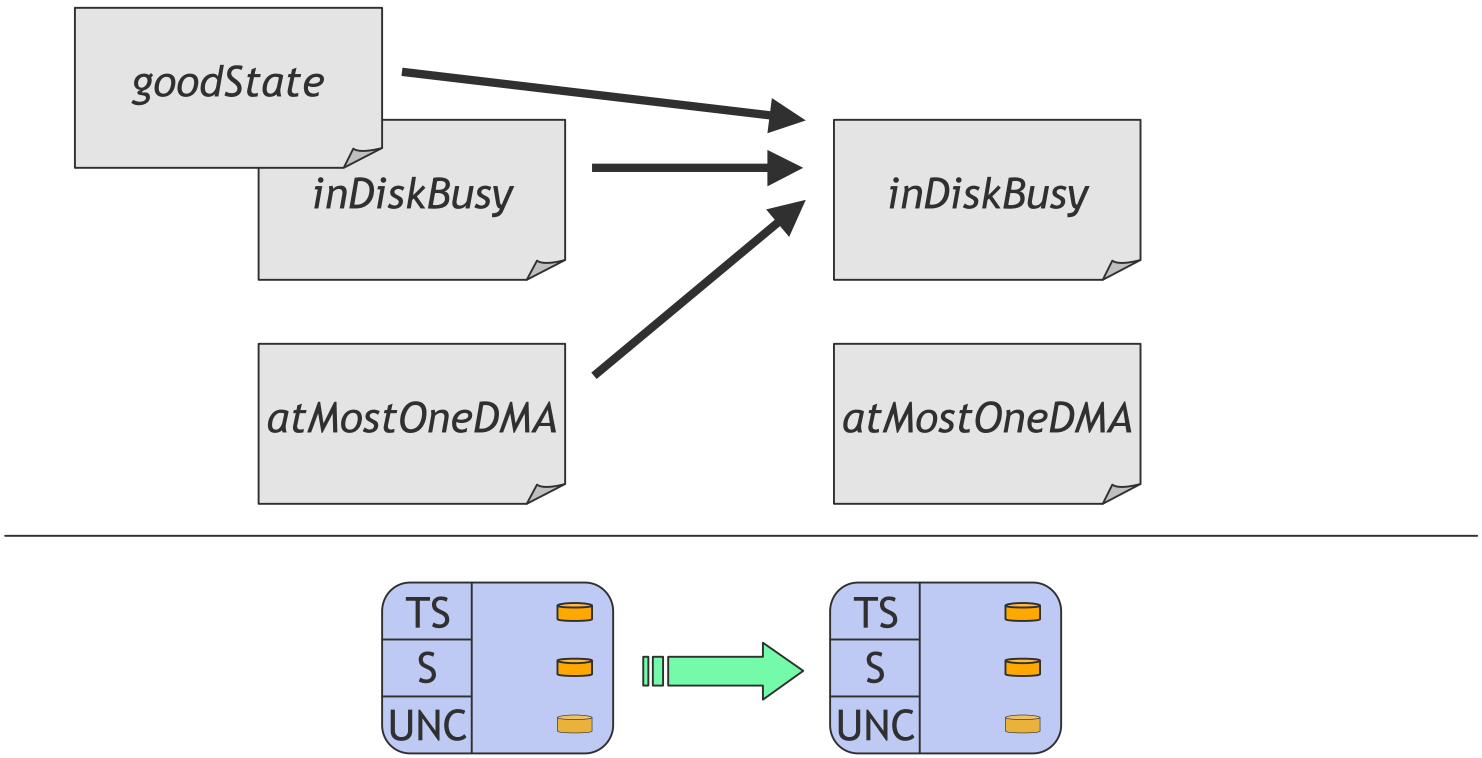
galois



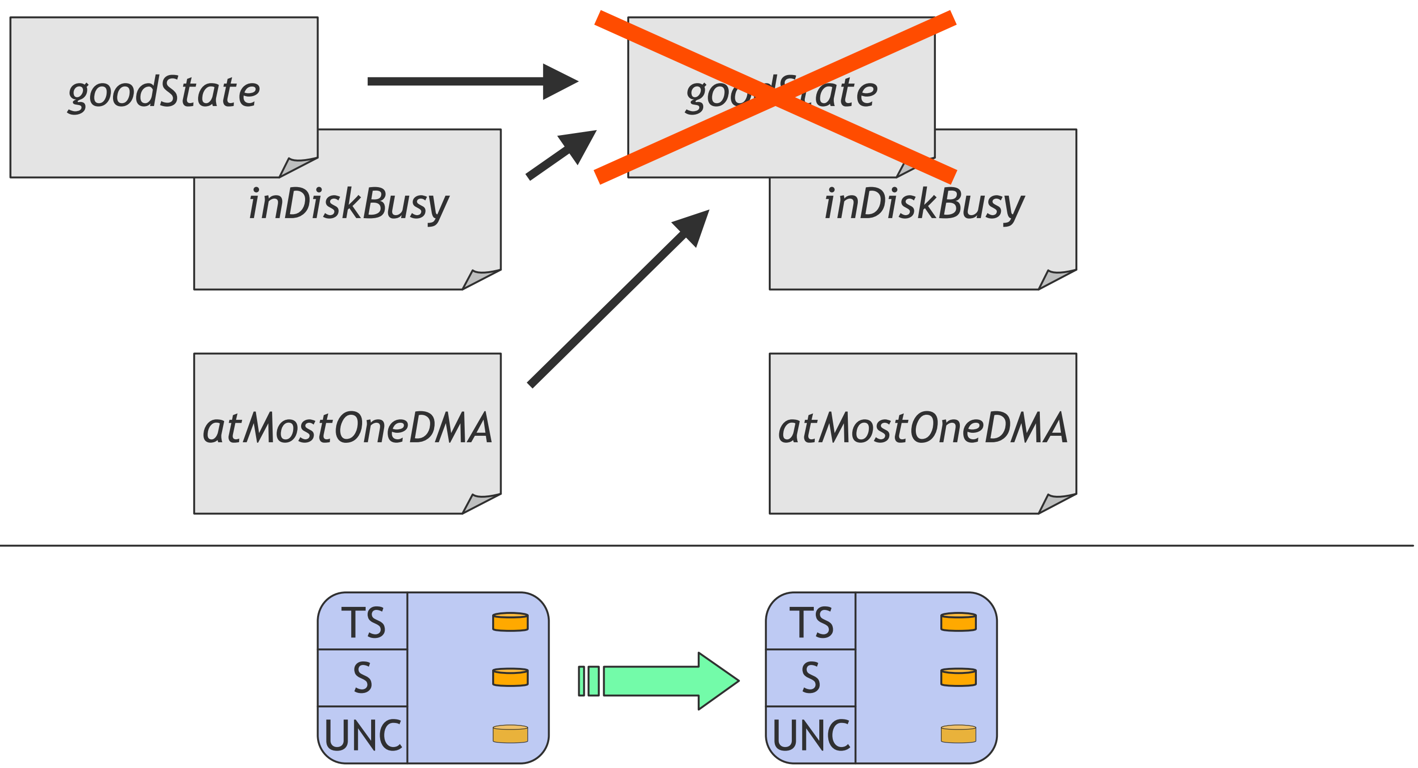
inDiskBusy
inDiskBusy
atMostOneDMA
atMostOneDMA
TS
S
UNC
TS
S
UNC

- Issue: we may have to go through many strengthening cycles before a strong enough invariant is found



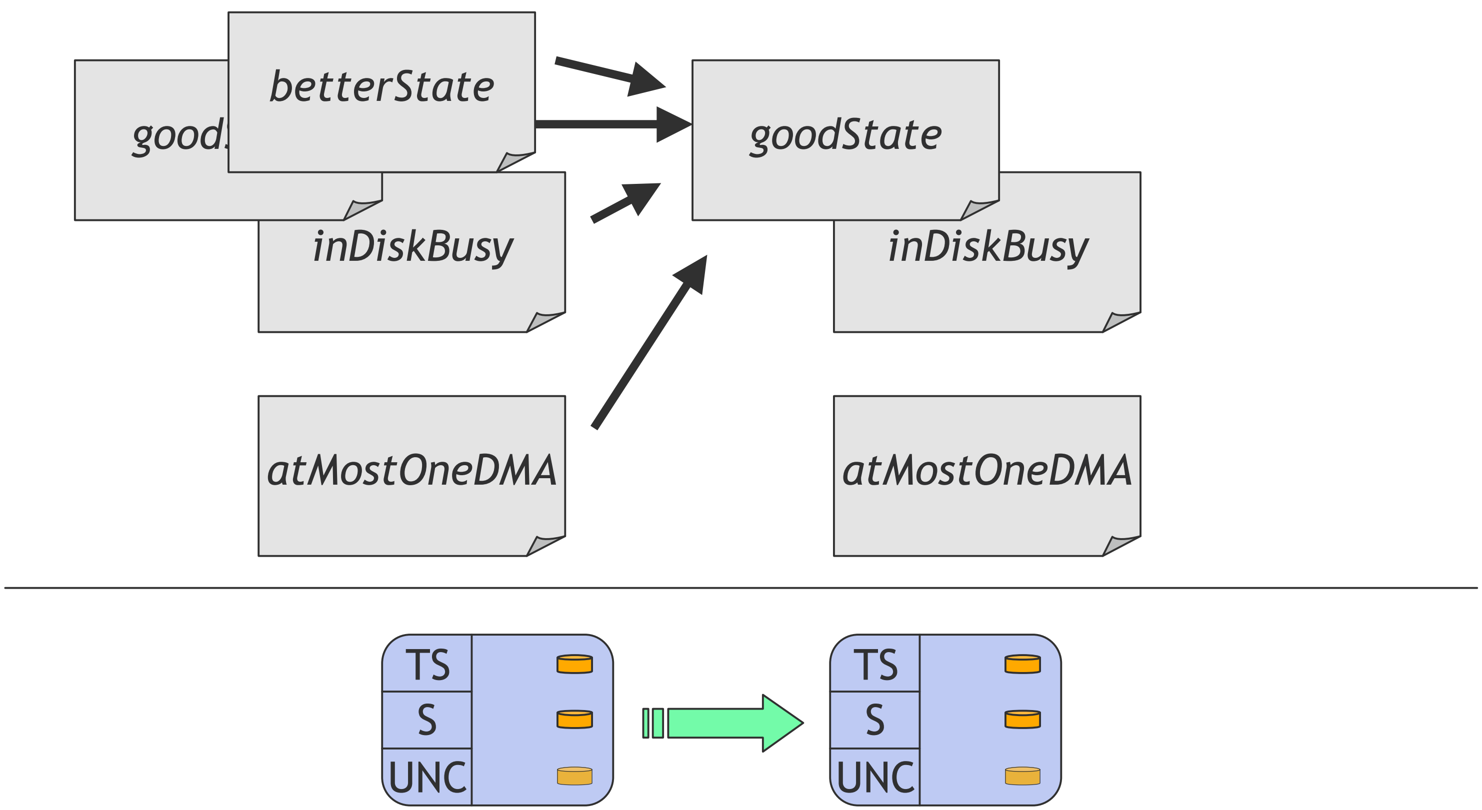

goodState
inDiskBusy
atMostOneDMA
inDiskBusy
atMostOneDMA
TS
S
UNC
TS
S
UNC

galois

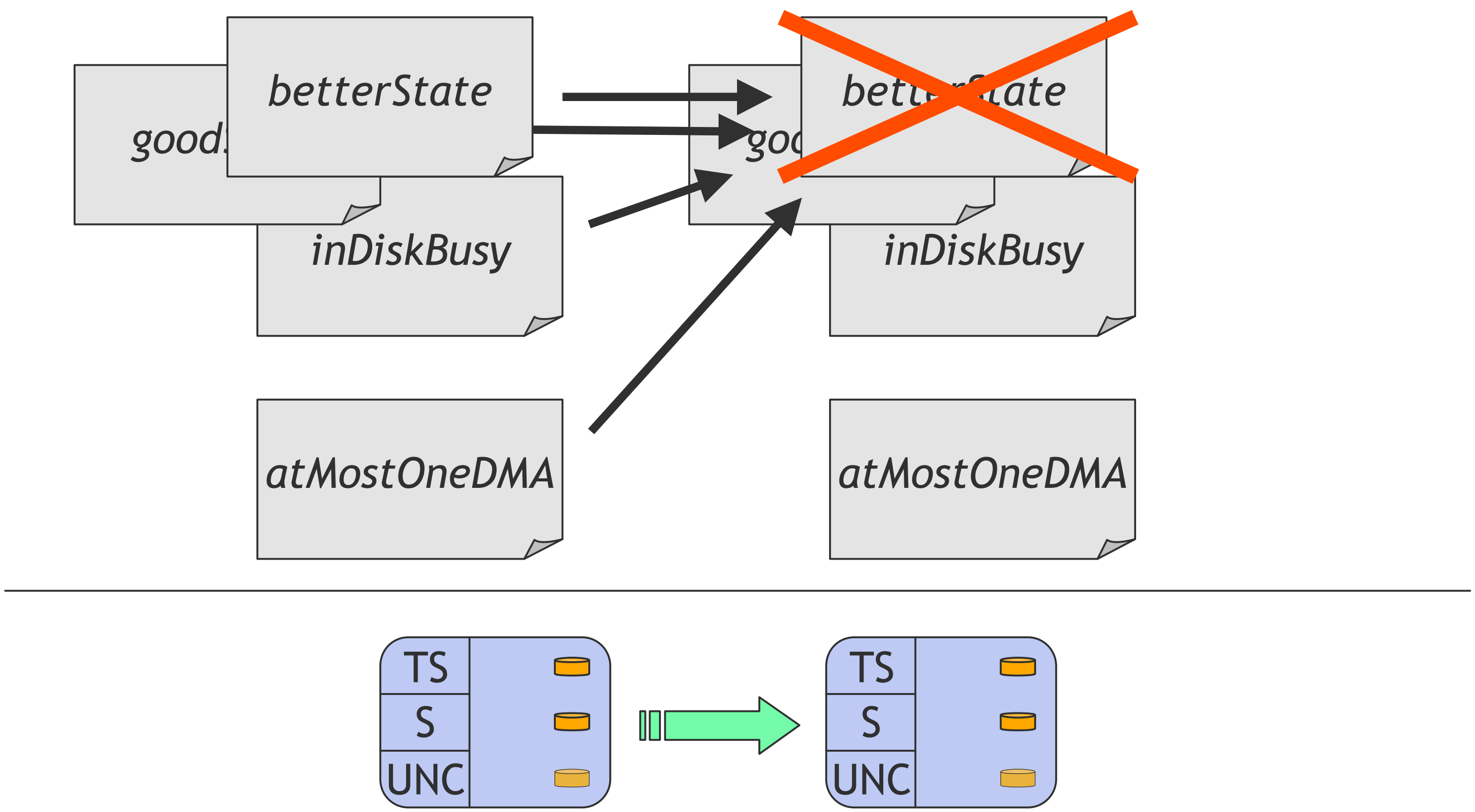


goodState
inDiskBusy
atMostOneDMA
inDiskBusy
atMostOneDMA
TS
S
UNC
TS
S
UNC

| galois |

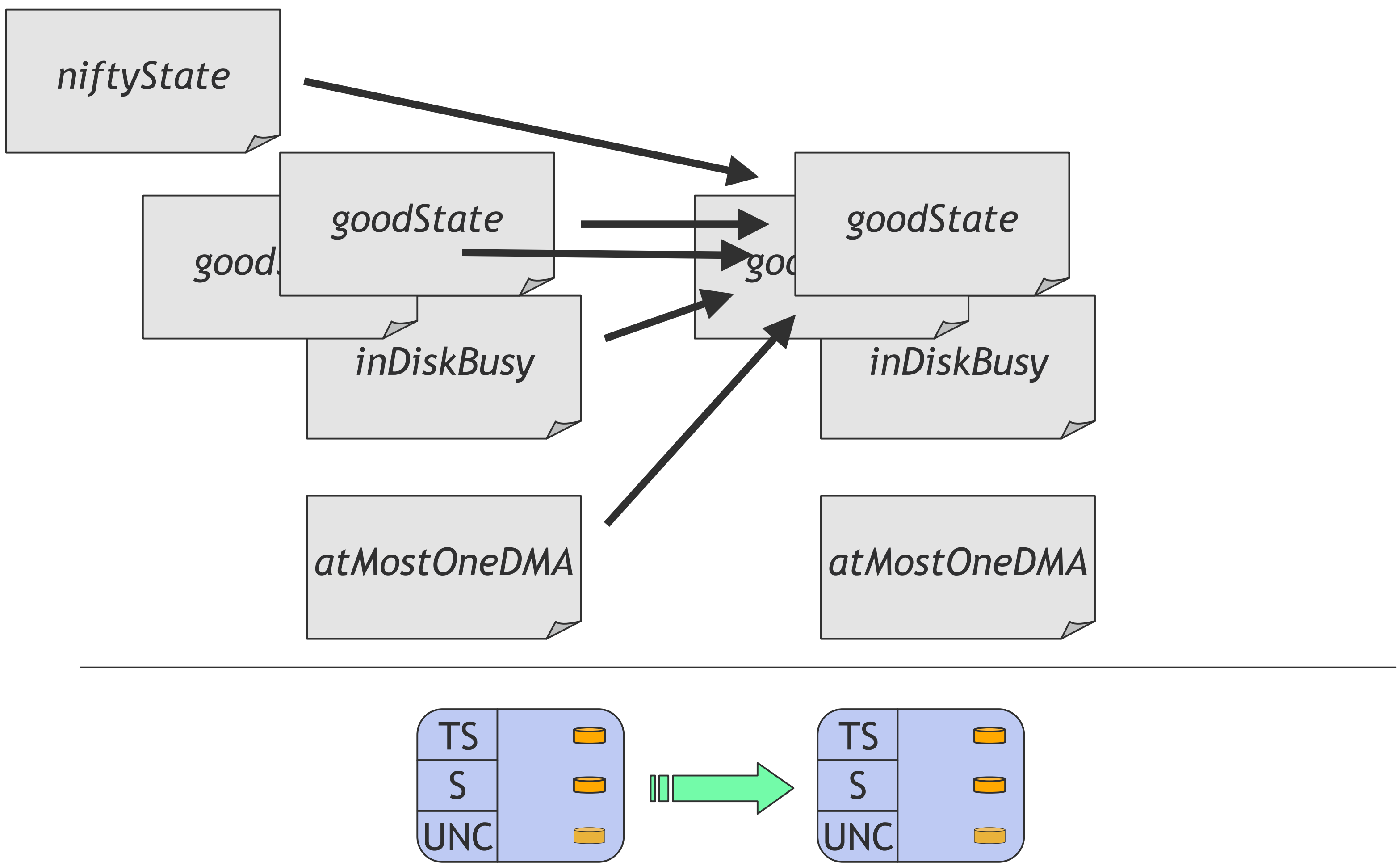

goodState
betterState
inDiskBusy
atMostOneDMA
goodState
inDiskBusy
atMostOneDMA
TS
S
UNC
TS
S
UNC

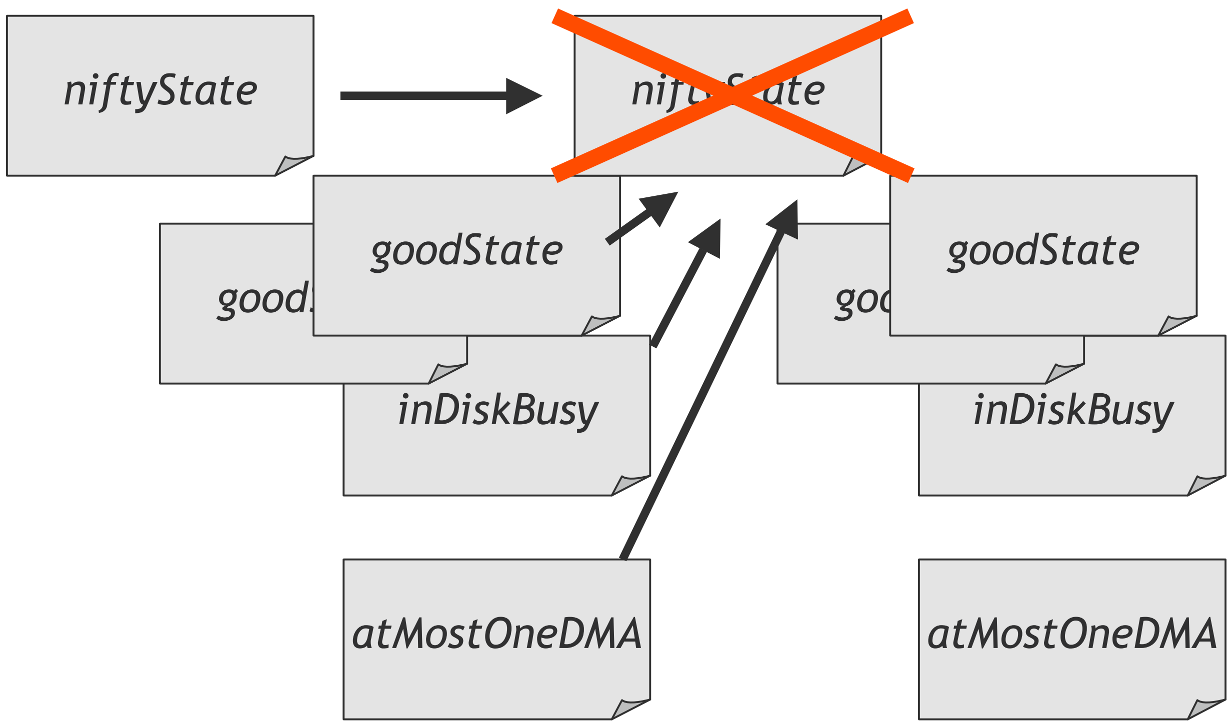
galois

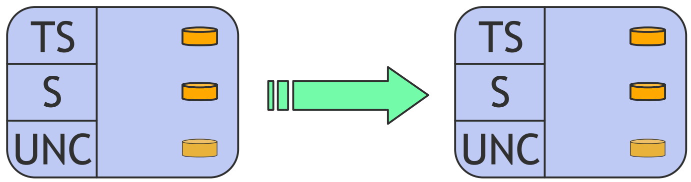


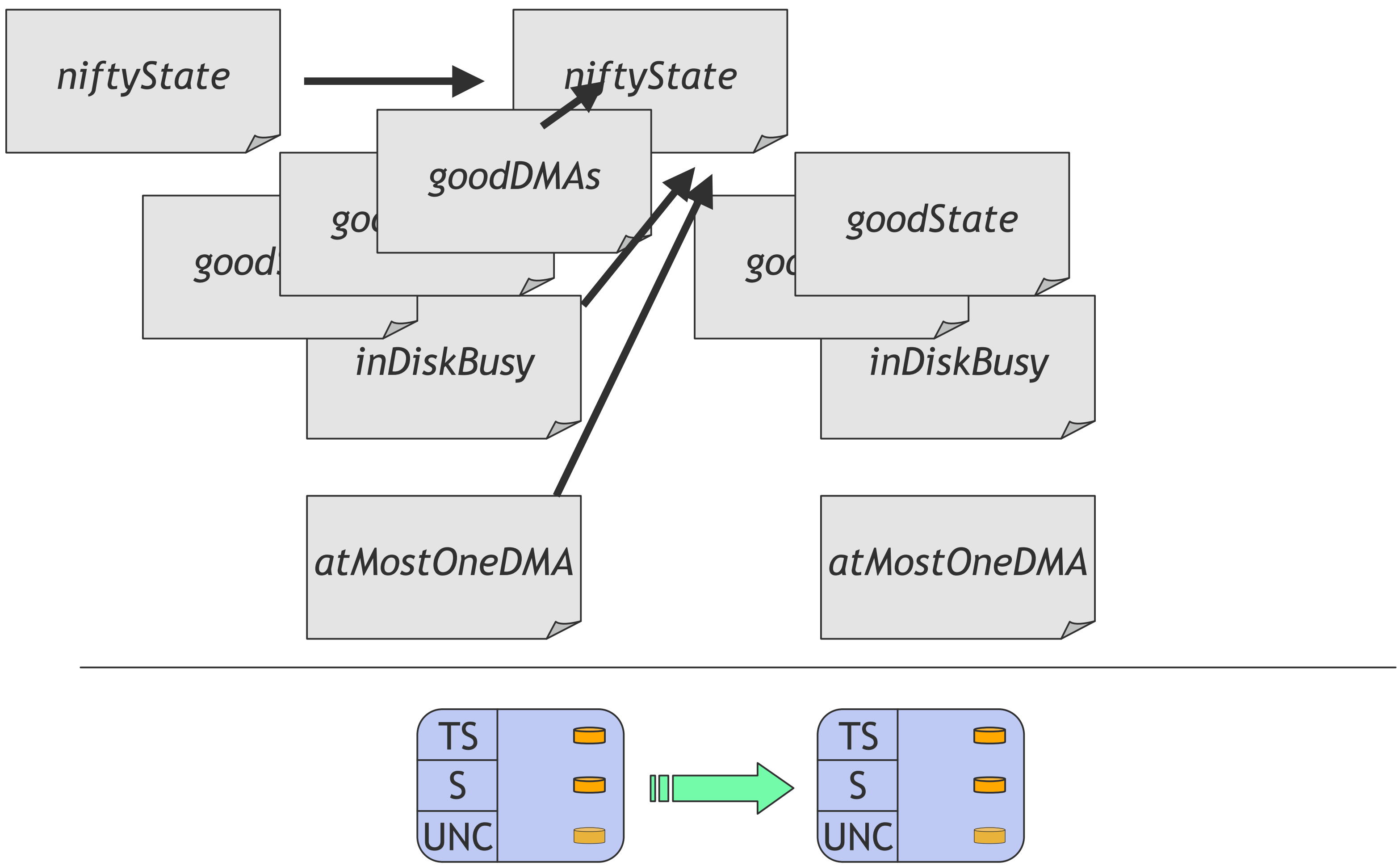
betterState
good
inDiskBusy
atMostOneDMA
betterState
inDiskBusy
atMostOneDMA
TS
S
UNC
TS
S
UNC

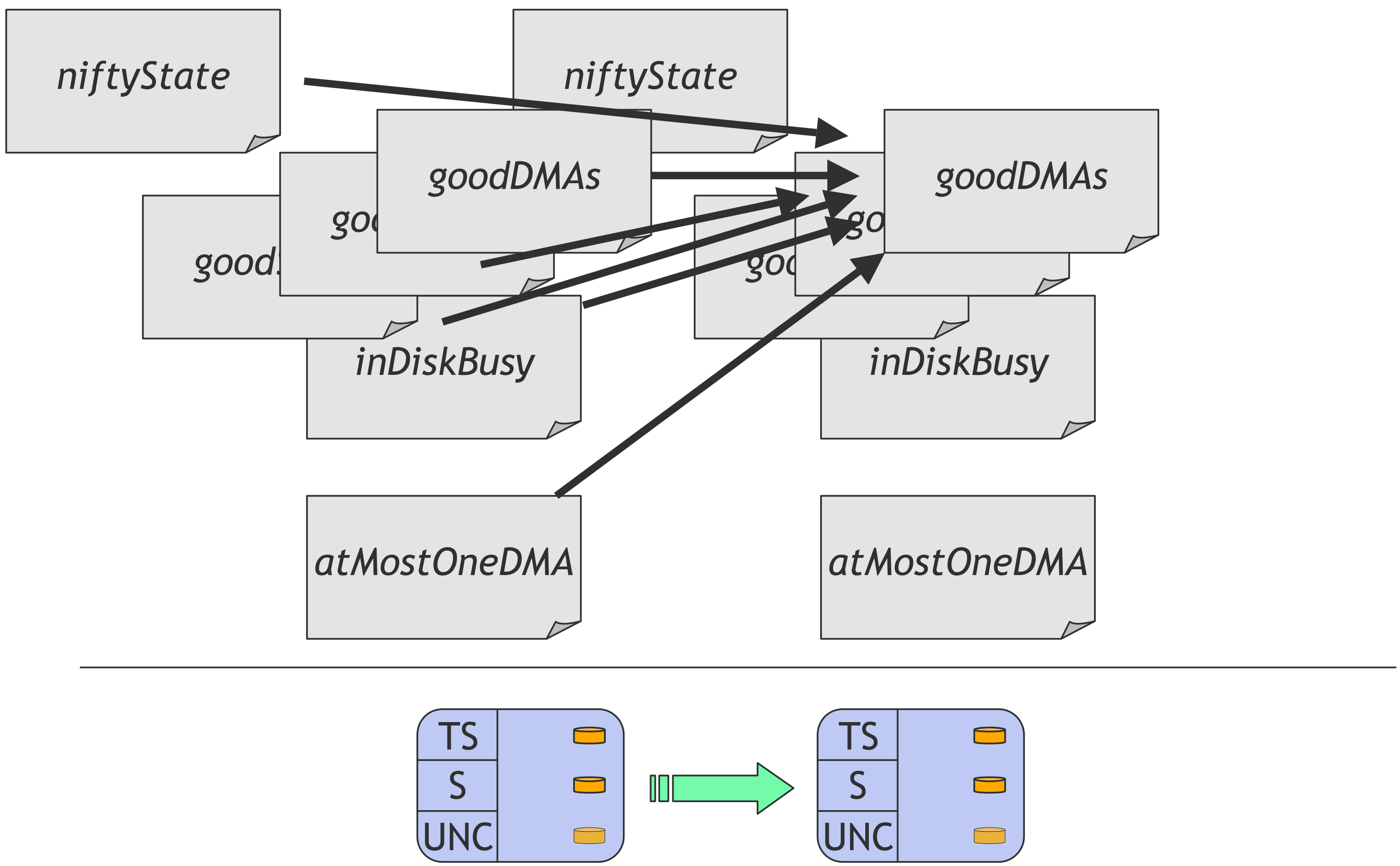
galois


niftyState
good
goodState
inDiskBusy
atMostOneDMA
go
goodState
inDiskBusy
atMostOneDMA
TS
S
UNC
TS
S
UNC


niftyState
niftyState
goodState
goodState
inDiskBusy
atMostOneDMA
goodState
goodState
inDiskBusy
atMostOneDMA

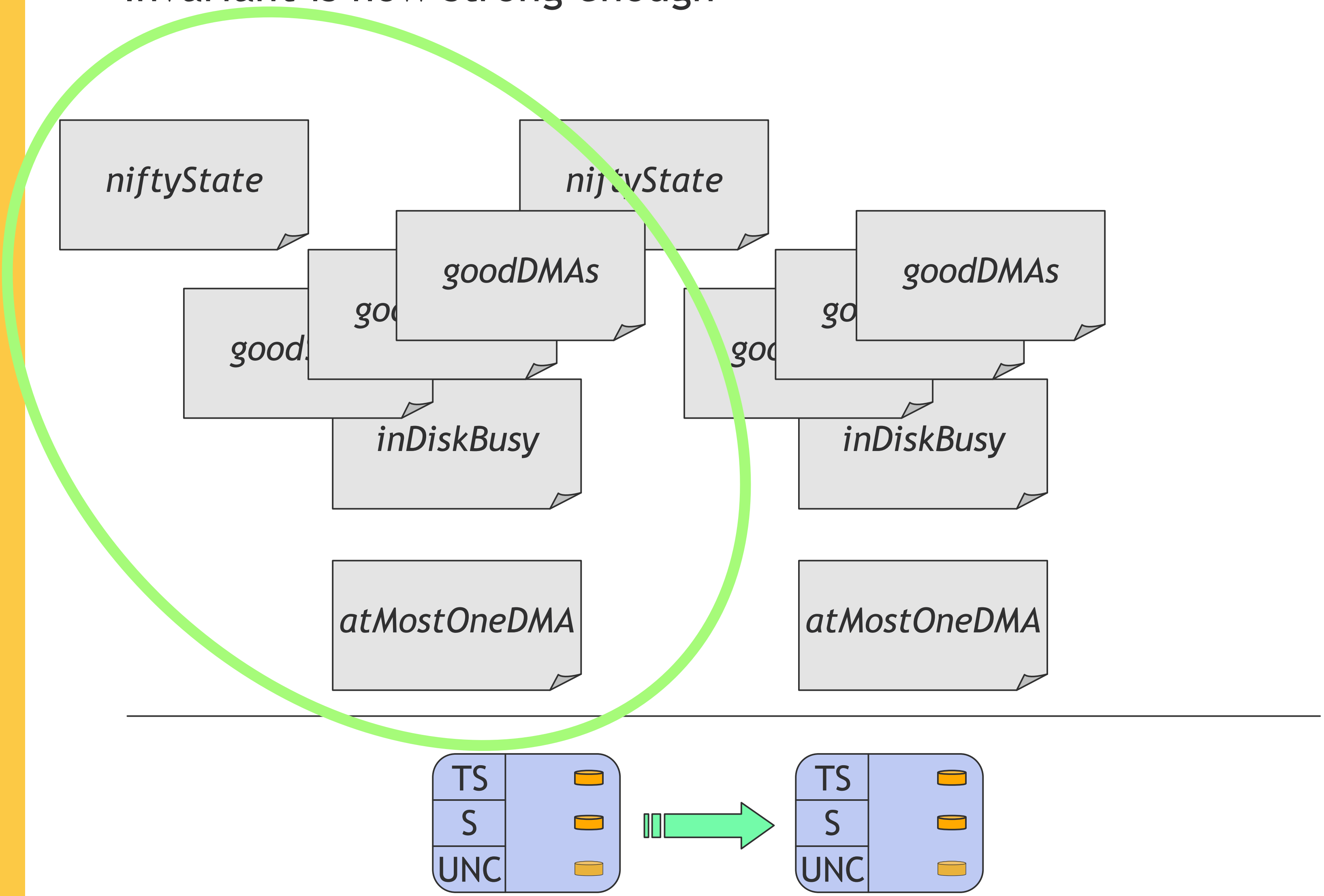

TS
S
UNC
TS
S
UNC


niftyState
niftyState
goodDMAs
goodState
inDiskBusy
inDiskBusy
atMostOneDMA
atMostOneDMA
TS
S
UNC
TS
S
UNC


niftyState
goodDMAs
inDiskBusy
atMostOneDMA
niftyState
goodDMAs
inDiskBusy
atMostOneDMA
TS
S
UNC
TS
S
UNC

galois

- Invariant is now strong enough



Theorem proving limitations when invariant strengthening

- Current theorem provers focus on machine-checking *correct* proofs
- Not enough support for debugging *incorrect* proofs
  - Isabelle doesn't provide any before-state and after-state counterexample information
  - We had to infer counterexample info by carefully examining how proof subgoals change during each step of the failed induction proof

Invariant strengthening is laborious!

Aug 1, 2005 (r3187)

about to extend goodState with the relationship between pending diskrequests, idle dma buffers, and read request continuations

Aug 11, 2005 (r3272)

I just need to handle startDma, pretty much. I looks like I need to **strengthen the goodState induction hypothesis**, which may break a lot of lemmas.

Aug 25, 2005 (r3342)

- **updated startDma invariant.**

Sep 9, 2005 (r3406)

strengthened induction hypothesis with goodIdle

Sep 26, 2005 (r3463)

- **strengthened induction hypothesis**

Oct 04, 2005 (r3495)

- updated dma completion to better match dma initiation
- about to **strengthen induction hypothesis** for dmaCompleteOk

Dec 19, 2005 (r3857)

- **changed <= to < in cont_set** for proper bounds checking

Invariant strengthening is laborious!

Dec 20, 2005 (r3862)
strengthened pending_set to insist on block sized transfers

Dec 21, 2005 (r3873)
strengthened invariant to (%s. s : state_set c Int busyInDiskOnce Int inDiskBusy)

Jan 3, 2006 (r3964)
- still **need to prove one additional invariant** (busyInDiskOnce) required by ProcessDisksSafety.thy

Mar 17, 2006 (r4748)
- strengthened safety invariant to include monotonicity of disk times

Mar 17, 2006 (r4753)
I need to **add** and propagate **a safety property** that the security levelsof the continuations match those of the pending dma requests.

Mar 20, 2006 (r4761)
propagated saftey constraint about equality of continuation and dma queue sizes

Apr 7, 2006 (r5017)
- **started establishing pendSup invariant** about the two traces used in non-interference

Apr 26, 2006 (r5197)
I still need to **compute the timing oracle** for the whole bacStep

…

galois

Outline

- *Tearline Wiki* system architecture
- Formally verifying the *Block Access Controller*
- **Making future verifications easier**

Software model checking

- We've successfully verified an 800 line cross-domain component
  - We need to scale this up to 10,000-line cross-domain components
- Can we leverage code analysis tools for this?
  - Code analyzers automatically strengthen loop invariants!
  - And generate a counterexample trace if the original invariant is false
- Example: SLAM software model checker
  - Statically checks that Windows device drivers maintain kernel state invariants
  - Has successfully checked drivers containing over 100,000 lines of C

Automated Security Analysis (ASA)

- ASA goal: Leverage existing code analyzers to check security properties of large C programs

- Starting to adapt open-source *Saturn* analyzer for checking information flow and buffer overrun properties
- Already finding vulnerabilities in open source security software
  - Neon 0.24.4: known format string vulnerability in XML 207 code
  - bftpd 1.6, smbftpd 0.96: unknown buffer underrun error in bftpd_stat (probably benign)
  - ISC DHCPD 3.0.1rc3: known format string vulnerability in print_dns_status. Other unknown but probably benign vulnerability.
  - cfengine 1.5.4: found two format string vulnerabilities (no false positives)

Code analysis tool limitations

- Code analyzers make simplifying assumptions. For example, SLAM assumes
  - No arithmetic overflow or underflow
  - Size of arrays = 1
- ASA project makes similar simplifying assumptions:
  - ```
    % XXX: it's really most interesting if the
    % trace refers to an argument, global, or return value.
    % If it only refers to locals, it's not as likely to be a
    % problem.
    ```
- Result: Code analysis *algorithms* are sound, but existing *tools* can be both unsound and incomplete.
  - Great for finding bugs in medium assurance code,
  - …but not for providing EAL7 assurance evidence
    ```

Software model checking limitations

- BAC state invariants contain many universal (∀) and existential (∃) quantifiers
  - Model checking quantified invariants is undecidable in general
  - Required manual quantifier instantiation steps in Isabelle proofs

- Examples of quantified BAC state invariants (discovered during invariant strengthening):
  - If a DMA is occurring to <u>any</u> memory page, then it is to a valid DMA buffer whose busy flag is set
  - If <u>any</u> DMA buffer's busy flag is set, then <u>there is a unique disk</u> that has a corresponding entry in its DMA queue
  - <u>For each</u> security level:
    - <u>The number of</u> pending DMA requests in memory to <u>any</u> disk is the same as <u>the number of</u> pending DMA requests on that disk.
    - <u>Each</u> DMA request in memory is to <u>some</u> disk at the same or lower security level

galois

Key research question

- How can we use decision procedures and code analysis algorithms in Isabelle to speed up invariant strengthening cycles?
  - While still allowing user to manually instantiate quantifiers when necessary

- Key benefit: provide EAL7 assurance evidence for much larger cross-domain components

First step: Isabelle SMT solver tactics

- Using an *SMT solver* to check invariants in Isabelle could really shorten invariant strengthening loops
  - SMT solvers are "push-button" decision procedures for a subset of first order logic
  - Can return before-state/after-state counter-example information when they can't prove the invariant
- Can still use "pure" Isabelle tactics to prove final strengthened invariant

`ismt` tactic

- **`ismt`** is an Isabelle *external oracle* we've developed for Yices
  - Yices: SMT solver developed at SRI

- Given a proof subgoal, **`ismt`**
  - Negates it,
  - Translates it to Yices' input language,
  - Calls Yices subprocess
    - UNSAT: Conclude theoremhood
    - SAT: Convert the model to a HOL counter-example

- Note: Isabelle automatically tracks all "Yices axioms" used in subsequent proofs

- We performed a preliminary experiment to see if **`ismt`** is helpful in proving invariants

Experiment: array copy

```
#define buf_size 32

int copy(int *src)
{
   int dst[buf_size];
   int *s = src, *d = dst;
   while(*s)
        *d++ = *s++;
   *d = 0;
   return 0;
}
```

Expanded/disambiguated program

```
#define buf_size 32

int copy(int *src)
{
   int dst[buf_size];
   int *s;
   int *d;
   s = src;
   d = dst;
   while(1)
     if(*s == 0)
       break;
     else
     {
       *d = *s;
       s++;
       d++;
       continue;
     }
   *d = 0;
   return 0;
}
```

Translation to monadic HOL

```
{
   int dst[buf_size];
   int *s;
   int *d;
   s = src;
   d = dst;
   while(1)
     if(*s == 0)
       break;
     else
     {
       *d = *s;
       s++;
       d++;
       continue;
     }
   *d = 0;
   return 0;
}
```

```
(doSeqC { with_array buf_size (λ(pdst :: int Ptr).
          with_var (λ(pps :: int Ptr Ptr).
          with_var (λ(ppd :: int Ptr Ptr). doSeqC {
          assign_ptr pps psrc;
          assign_ptr ppd pdst;
          loopAsrt
            (loopInv False psrc pdst pps ppd buf_size)
            (loopInv True psrc pdst pps ppd buf_size)
            (λ r s. False)
            (doSeqC {ps ⟵ deref_ptr pps;
                     ct ⟵ deref_ptr ps;
                     if (ct = 0)
                     then break
                     else doSeqC {pd ⟵ deref_ptr ppd;
                                  assign_ptr pd ct;
                                  assign_ptr pps (ps +p 1);
                                  assign_ptr ppd (pd +p 1);
                                  continue}});
          pd ⟵ deref_ptr ppd;
          assign_ptr pd 0;
          c_return 0
         })))
})"
```

Verifying the loop invariant

- Formalized a monadic Hoare logic and wrote a verification condition generator (VCG) tactic in Isabelle

- Isabelle simplifier and **`ismt`** tactic called on each verification condition in **`copy`** procedure
  - We first fixed the size of each array
  - **`ismt`** returned counterexample info each time invariant (or precondition) was too weak
  - **`ismt`** calls succeeded once invariant was strong enough

Final strengthened loop invariant

```
definition
  loopInv :: "bool ⇒ int Ptr ⇒ int Ptr ⇒
              int Ptr Ptr ⇒ int Ptr Ptr ⇒
              C_size ⇒ C_heap ⇒
              bool" where
  "loopInv aboutToBreak psrc pdst pps ppd sz s =
     (let h     = heap s;
          st    = status s;
          vpsrc = to_void_ptr psrc;
          vpdst = to_void_ptr pdst;
          vpps  = to_void_ptr pps;
          vps   = fromByte (h vpps);
          vppd  = to_void_ptr ppd;
          vpd   = fromByte (h vppd);
          bytes_copied = vps - vpsrc
      in  (if aboutToBreak
           then ( mem_inited vppd 1 st
                ∧ mem_alloced vpd 1 st)
           else ( distinct ( [vpps, vppd]
                          @ null_byte_span vps sz h
                          @ int_span vpdst sz)
                ∧ mem_inited vppd 1 st
                ∧ mem_inited vpps 1 st
                ∧ mem_alloced vpdst sz st
                ∧ vpsrc ≤ vps ∧ vps < vpsrc + sz
                ∧ vpd = vpdst + bytes_copied
                ∧ null_terminated_block_lim vps
                    (sz - bytes_copied) sz s)))"
```

Current status

- Fully automatic **`copy`** memory safety proof for fixed array size
- Currently proving **`copy`** memory safety for arbitrary array sizes
  - Requires quantified loop invariant
- Finding out how helpful "abstract" counterexample information is in finding quantifier instantiations
  - Adding instantiated formulas interactively when calling **`ismt`**
- Preliminary results:
  - Abstract counterexamples do help in finding quantifier instantiations
  - But dozens of instantiations are needed
  - Most instantiations are actually rewrite rules for functions that Yices doesn't know about

Next steps

- Incorporate rewriting directly into SMT solver
  - Solver could then interpret domain-specific functions
- Isabelle *theory solver* tactics
  - Called repeatedly as SMT solver explores partial models
  - Each call returns either
    - Theorem saying partial model is inconsistent -- SMT solver prunes that part of search space.
    - Concrete witness that model is satisfiable.
    - Zero or more new derived facts.
- These require custom SMT solver extensions
  - So we're also starting to use Intel's *Decision Procedure Toolkit* (DPT), an open source SMT solver

Conclusions

- Code analyzers unlikely to provide EAL7 assurance
  - Most analyzers make unsound simplifying assumptions
  - Cross-domain components have quantified state invariants
- Theorem provers can provide EAL7 assurance for small cross domain components
  - Took one engineer-year to verify 800-line BAC
- Reducing cost of formal verification is essential to scale up EAL7 assurance
  - Greatest TSE project risk was BAC verification
  - Integrating code analysis algorithms into Isabelle could help a lot
- We're pursuing an open source strategy
  - Galois is too small to fund this "infrastructure" project through IR&D

Questions