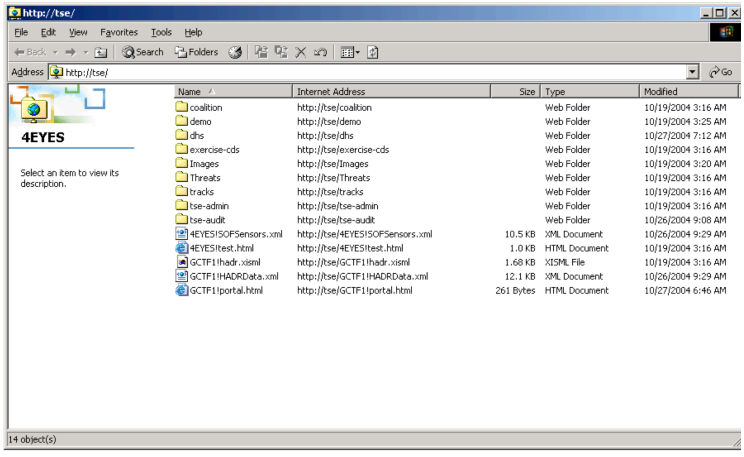
Cross-Domain WebDAV Server

John Launchbury
CUFP 2007

galois

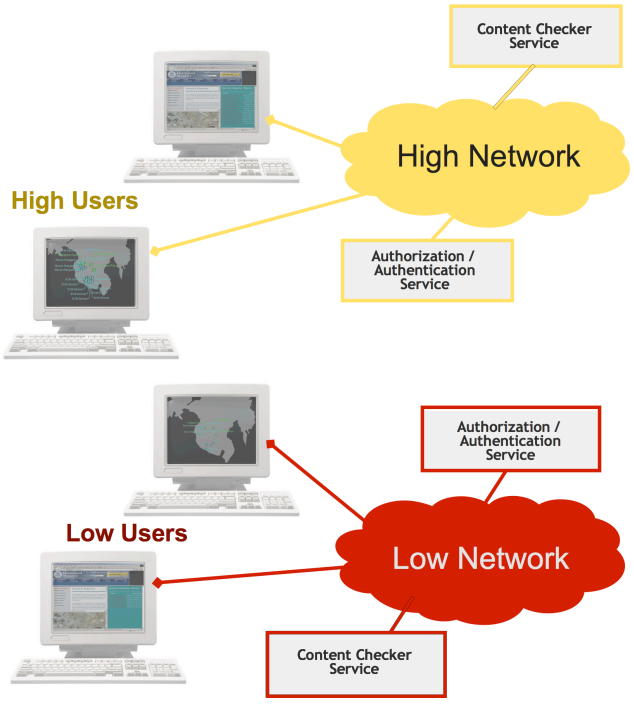
WebDAV the protocol

- WebDAV: "Web-based Distributed Authoring and Versioning"
  - Standard internet protocol (www.webdav.org)
  - HTTP gives read access over a network:
    WebDAV gives write access also
- Any media type
  - HTML, JPEG, MS Office, etc.
- Designed for enterprise collaboration
- Already in Windows etc.



galois

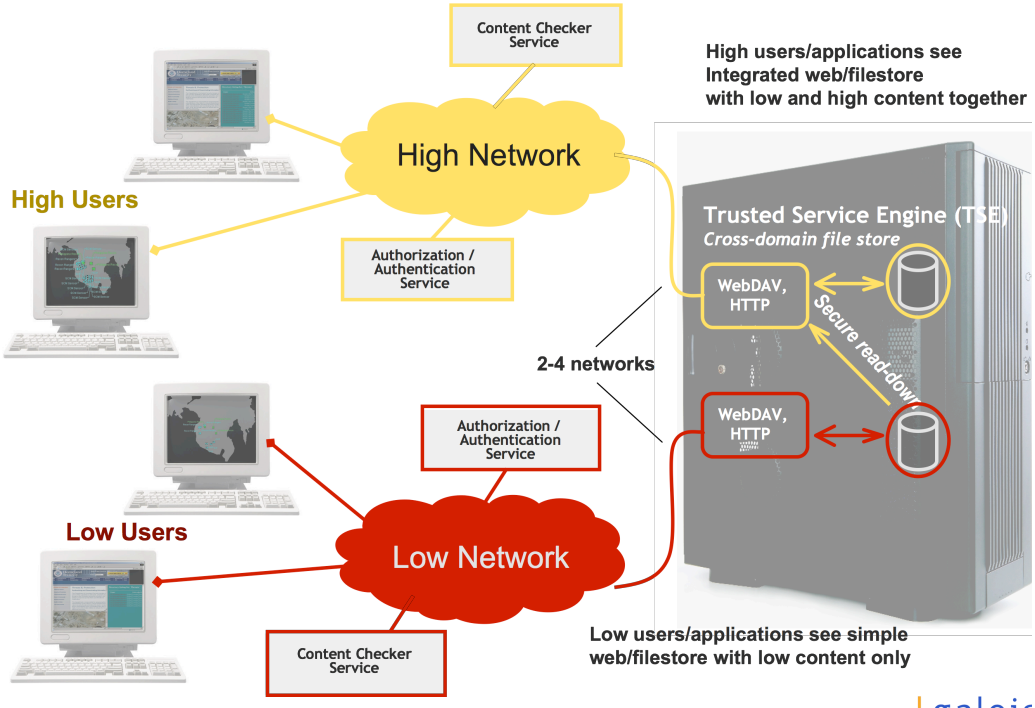
Cross-Domain WebDAV Server: TSE



- Separate networks are used to separate information at different classifications levels
- These networks are air-gapped to prevent information leakage

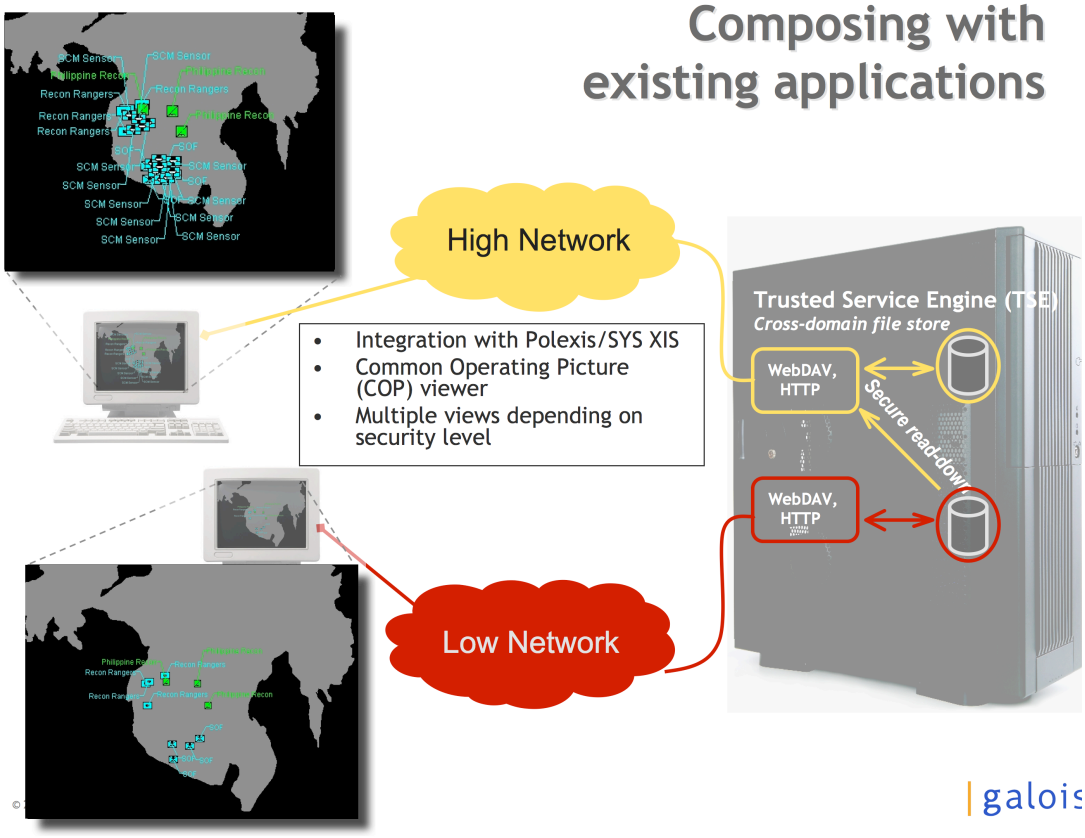
galois

Cross-Domain WebDAV Server: TSE



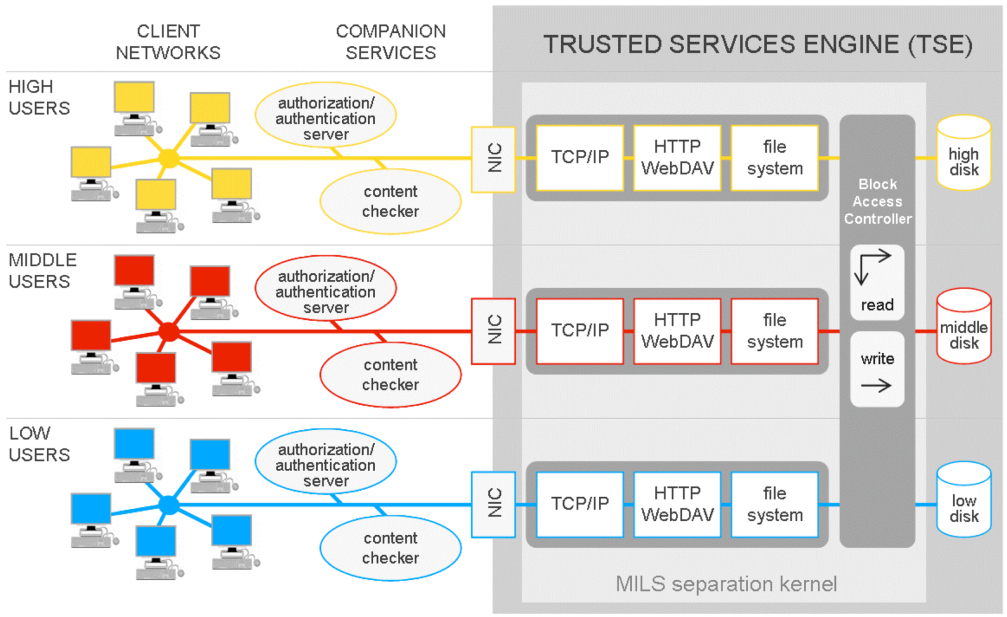
galois

Composing with existing applications



galois

TSE Architecture

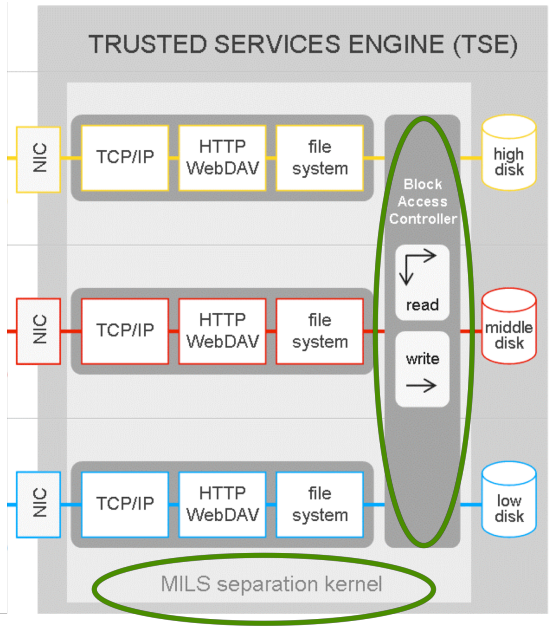


galois

TSE Architecture

MILS = Multiple Independent Levels of Security

1. Factor the security architecture
2. Minimize the number of components requiring high assurance
3. Keep each as simple as possible
4. Use formal methods in critical places



galois

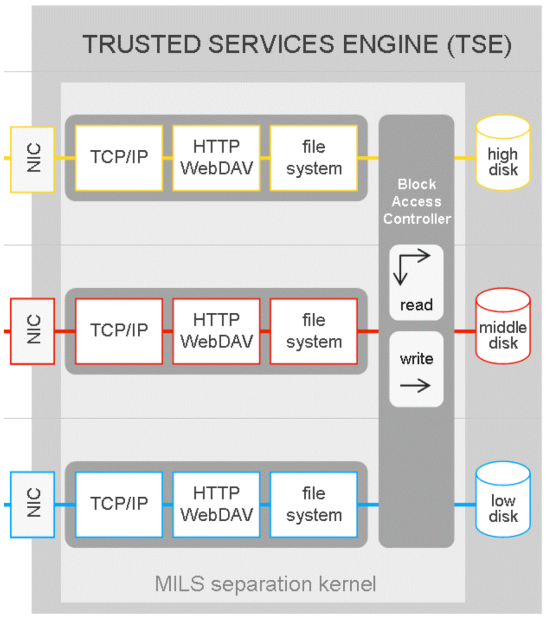
MILS for secure network servers

- MILS is an architecture, not an operating system
  - The OS provides separation only
    - Higher level system features are provided by user-level application processes
  - In contrast to systems built on “trusted OS’s”
    - MILS systems can have a smaller TCB (trusted computing base)
    - They provide less features to applications
  - MILS is near deployment in embedded use
- MILS separation kernels
  - Green Hills' INTEGRITY OS for x86 platforms
  - SE Linux, configured as a separation kernel
- The TSE project has broken ground in the use of MILS for secure network servers
  - Demonstrated MILS is feasible for servers
  - Contributed an approach for DMA to the community
  - Shows that MILS can be a key infrastructure where web services have cross-domain requirements

galois

File System Characteristics

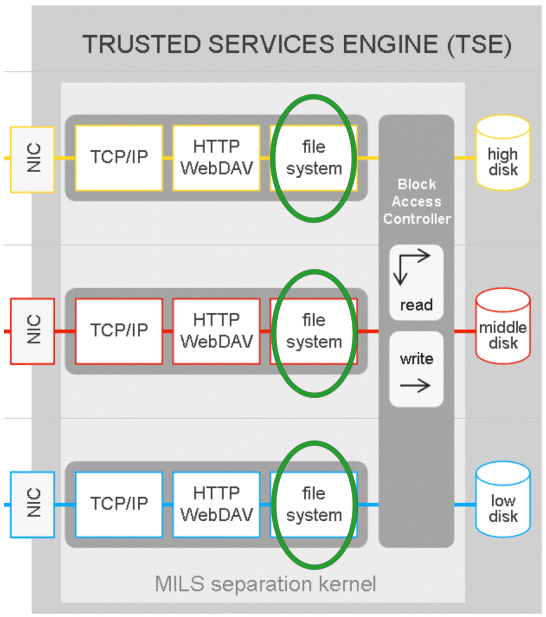
- Single-level file systems access multiple disk drives
  - Drives and their firmware are outside the TCB
- Read access from high must be invisible to low
  - No locking (hi-lo channel)
  - No abort/retry (lo-hi denial of service)
- No existing file system would work
  - Traditional cache coherency is infeasible across security boundaries
  - Designed WFFS (wait-free file system)



galois

File System Characteristics

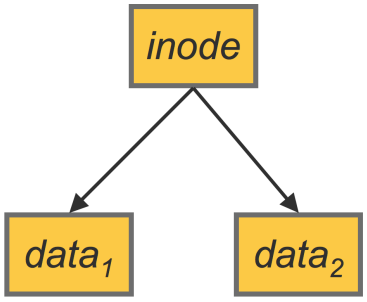
- Single-level file systems access multiple disk drives
  - Drives and their firmware are outside the TCB
- Read access from high must be invisible to low
  - No locking (hi-lo channel)
  - No abort/retry (lo-hi denial of service)
- No existing file system would work
  - Traditional cache coherency is infeasible across security boundaries
  - Designed WFFS (wait-free file system)



galois

WFFS: Wait-free synchronization example

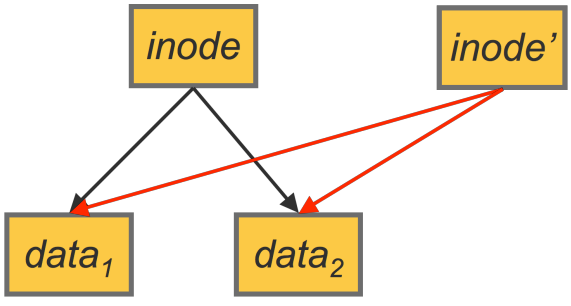
- Writer updating a file



WFFS: Wait-free synchronization example

- Writer updating a file

Open *copies inode*

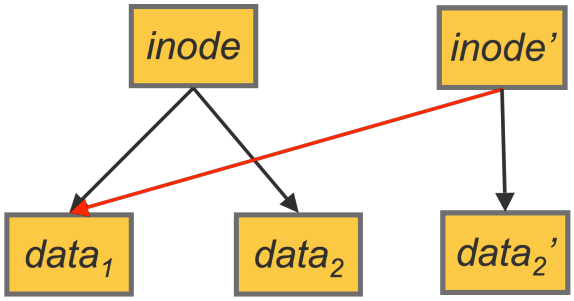


WFFS: Wait-free synchronization example

- Writer updating a file

Open *copies inode*

Write *copies & installs new blocks*



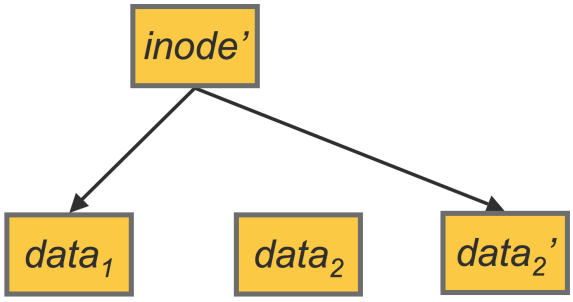
WFFS: Wait-free synchronization example

- Writer updating a file

Open *copies inode*

Write *copies & installs new blocks*

Close *installs inode'*

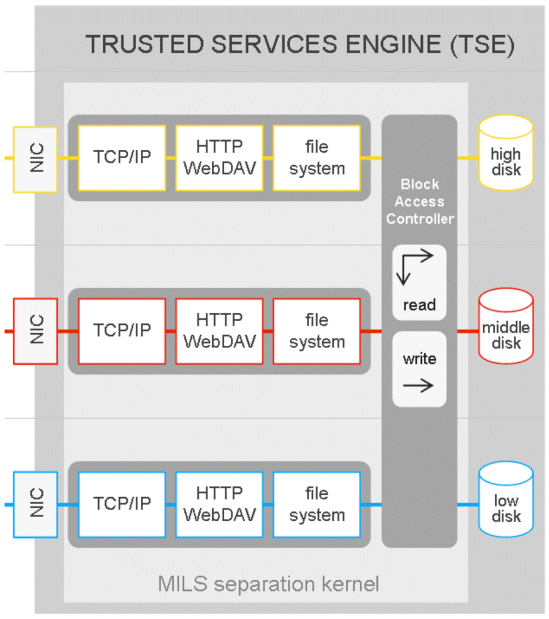


WFFS Consistency properties

1. Disk reads from High get latest written consistent inodes & data
2. Low is isolated from High's actions
   - Neither reads nor writes are affected by high reads
3. Cached contents of blocks and inodes are valid even as Low over-writes & deletes files
   - How frequently to synchronize cache between high and low is a policy choice
4. Low can safely reuse a block referenced by High
   - FreeQOut signal written to disk
   - High detects and flushes read only cache

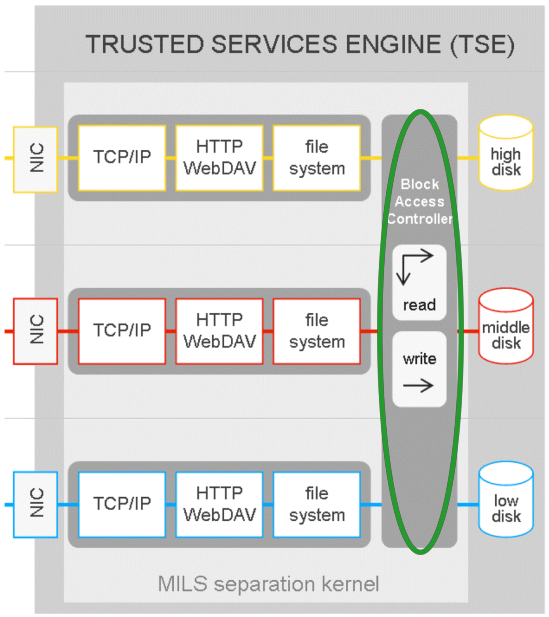
Block Access Controller

- BAC directs accesses across multiple disk drives
  - Block-level requests
  - Request queuing
- BAC spans partitions at multiple levels
  - High assurance component
  - Eliminate data channels between levels
  - Control timing channels between levels
- Written in simple C !
  - About 1000 lines



Block Access Controller

- BAC directs accesses across multiple disk drives
  - Block-level requests
  - Request queuing
- BAC spans partitions at multiple levels
  - High assurance component
  - Eliminate data channels between levels
  - Control timing channels between levels
- Written in simple C !
  - About 1000 lines

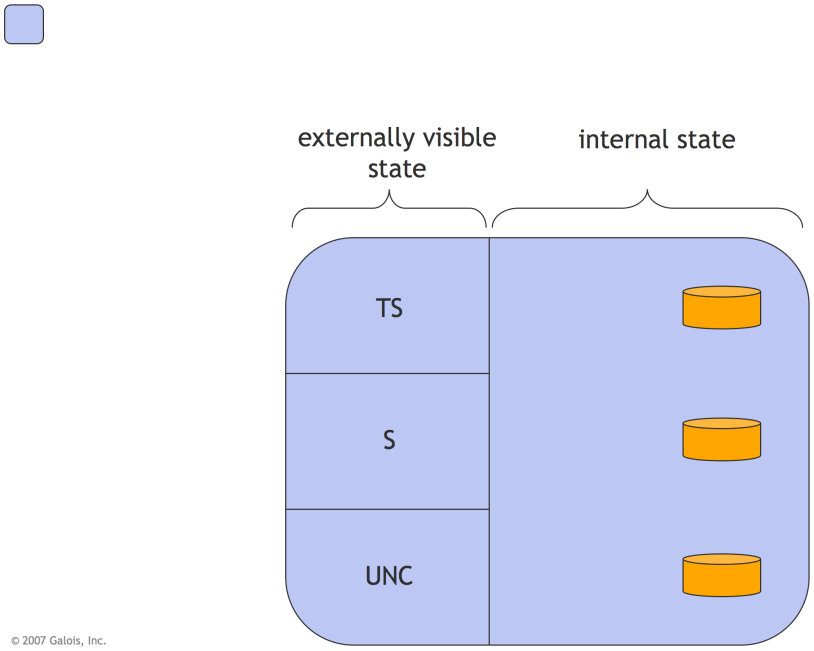


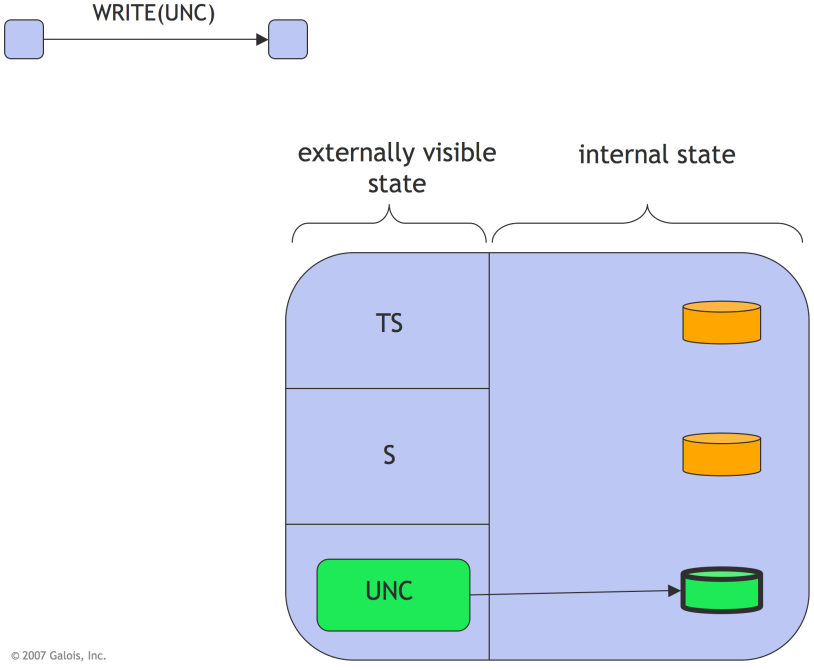
galois

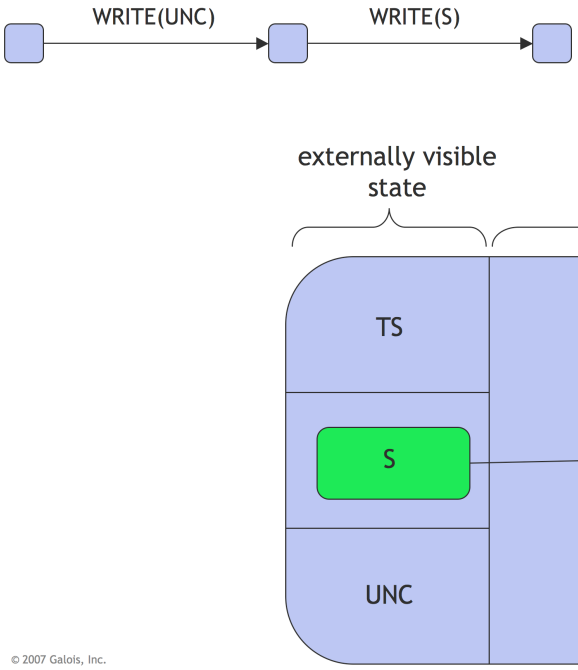
BAC Security Properties

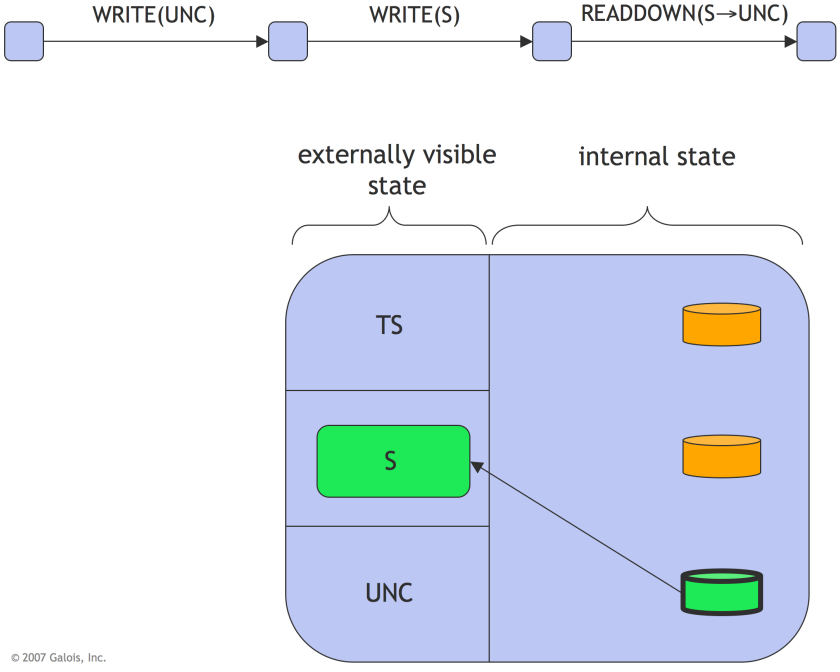
- No error states can occur, e.g.
  - Out-of-bounds array access
  - Out-of bounds disk block ID
  - Access to memory undergoing DMA transfer
  - Multiple simultaneous DMA transfers to same memory region
  - Too many simultaneous DMA transfers to a single disk
- A partition's state is not dependent on the actions of any higher level partition
  - Including number of partition time-slices it takes the BAC to respond to a request
  - Formalize a variation of von Oheimb's theory of noninterference in Isabelle
  - Prove noninterference for a model of the C-code

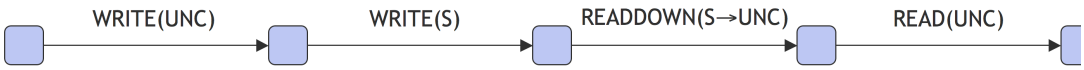
galois

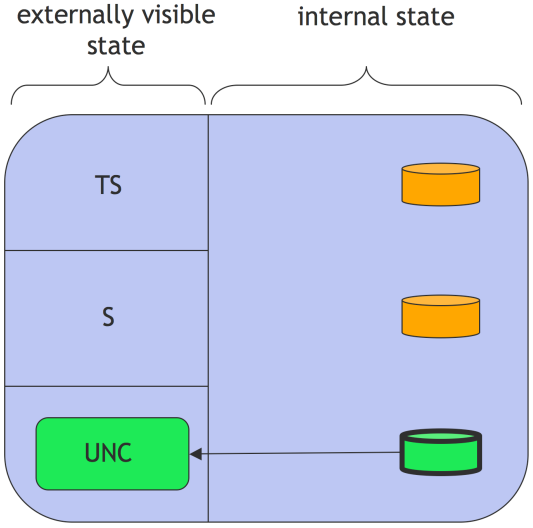




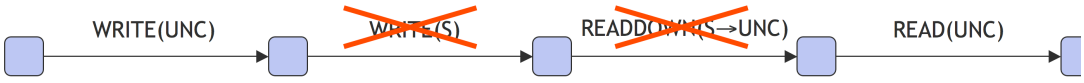


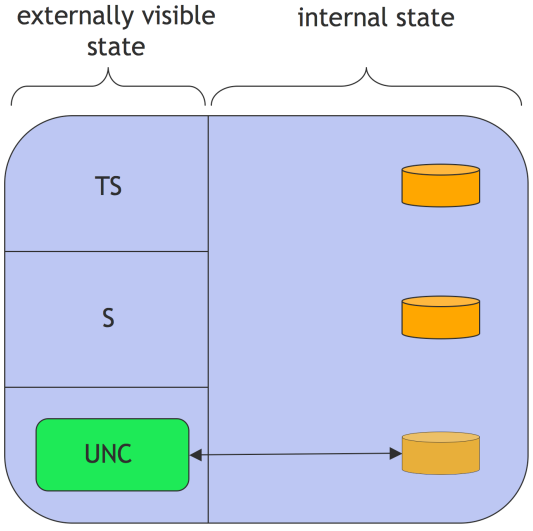






galois





Non-interference

- Actions at higher security levels not externally visible current security level
- Prove that *purging* higher security level actions from event trace result in same externally visible state

galois

Model-to-code correspondence (conventional)

```
responseSet (level::level) (resp::response) (i::nat)
            (inp::input) (st::state)
 = let respN = responseOvec level;
       x0 = bytesPerResponseRep * i
   in (case ovecRef respN x0 inp st of
         Error => Error
       | Ok (st2,oldToggle) =>
          (case ovecSet respN (1 + x0) (respOk resp) inp st2 of
             Error => Error
           | Ok (st3,u) =>
               ovecSet respN x0 (toggleNat oldToggle) inp st3))
```


Model-to-code correspondence (conventional)

```
responseSet (level::level) (resp::response) (i::nat)
            (inp::input) (st::state)
 = let respN = responseOvec level;
       x0 = bytesPerResponseRep * i
   in (case ovecRef respN x0 inp st of
         Error => Error
       | Ok (st2,oldToggle) =>
          (case ovecSet respN (1 + x0) (respOk resp) inp st2 of
             Error => Error
           | Ok (st3,u) =>
               ovecSet respN x0 (toggleNat oldToggle) inp st3))
```

```
void responseSet (level level, response resp, nat i) {
  nat respN = responseOvec(level);
  nat x0 = bytesPerResponseRep * i;
  nat oldToggle = ovecRef(respN, x0);
  ovecSet(respN, 1 + x0, respOk(resp));
  ovecSet(respN, x0, toggleNat(oldToggle));
}
```

Model-to-code correspondence (conventional)

```
responseSet (level::level) (resp::response) (i::nat)
            (inp::input) (st::state)
 = let respN = responseOvec level;
       x0 = bytesPerResponseRep * i
   in (case ovecRef respN x0 inp st of
         Error => Error
       | Ok (st2,oldToggle) =>
          (case ovecSet respN (1 + x0) (respOk resp) inp st2 of
             Error => Error
           | Ok (st3,u) =>
               ovecSet respN x0 (toggleNat oldToggle) inp st3))
```

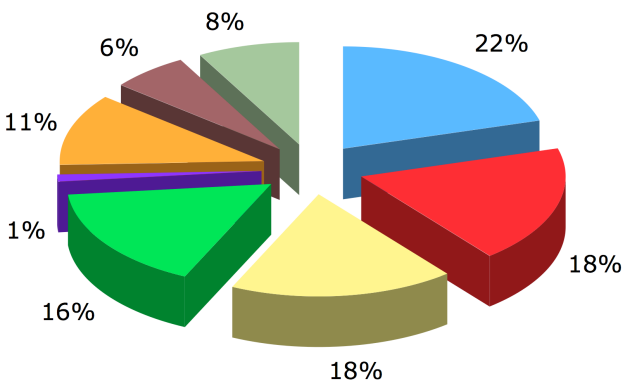
```
void responseSet (level level, response resp, nat i) {
  nat respN = responseOvec(level);
  nat x0 = bytesPerResponseRep * i;
  nat oldToggle = ovecRef(respN, x0);
  ovecSet(respN, 1 + x0, respOk(resp));
  ovecSet(respN, x0, toggleNat(oldToggle));
}
```


Model-to-Code Correspondence (Monadic)

```
responseSet (level::level) (resp::response) (i:nat)
 = let respN = responseOvec level;
       x0 = bytesPerResponseRep * i
   in do oldToggle <- ovecRef respN x0;
         ovecSet respN (1 + x0) (respOk resp);
         ovecSet respN x0 (toggleNat oldToggle)
```

```
void responseSet (level level, response resp, nat i) {
  nat respN = responseOvec(level);
  nat x0 = bytesPerResponseRep * i;
  nat oldToggle = ovecRef(respN, x0);
  ovecSet(respN, 1 + x0, respOk(resp));
  ovecSet(respN, x0, toggleNat(oldToggle));
}
```

Line Count Breakdown of TSE



Total: 54KLOC

galois

Haskell

- Heap and RTS
  - Prohibitively complex for creating secure components
- Flexibility of Haskell enable substantial changes
  - Authentication and a plugin architecture added after the TSE was essentially finished as web server
  - Multi-disk buffer block cache added to our file system after the file system was complete
- FFI worked well and reliably
  - The low-level access to the BAC
  - The interace to the SSL library written in C
- Modules system mostly worked well
  - Would have liked a way to manage imports more flexibly

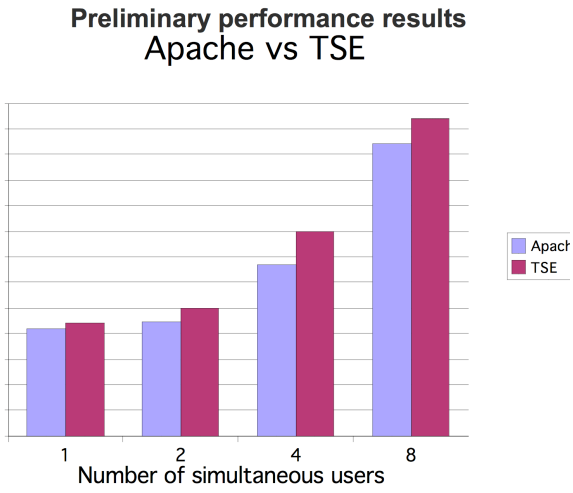
galois

Great Haskell Tools

- Profiler guided all the performance improvements
- Testing by both QuickCheck and HUnit
  - Many tests were IO based, used lots of HUnit tests
  - HPC developed to address coverage needs; not yet used in detail
- Tried to use Cabal to put multiple packages into a system hierarchy
  - TSE had around a dozen packages
  - Had to make a complex build system that had to figure out dependencies
  - Also had to customize Cabal also to allow build-local packages
- Haddock
  - Haddock lets you add comments to types, generating HTML docs
  - Worked beautifully!

Performance

- Laziness
  - "To a first approximation, strictness versus laziness didn't matter squat." Andy Gill, key developer
- Most early performance problems were with manipulation of binaries
  - Original bhPut copied Binary objects byte-by-byte
  - Modified version used clib's memmove



Conclusion

- Haskell gave us the engineering freedom to build the system right
  - Implement big new capabilities like the file-system
  - Performance is great
  - Concurrency was really easy to use
  - Straightforward to enable interaction with non-Haskell parts
- Big systems could benefit from better Haskell infrastructure
  - Flexible module import
  - Compilation manager
- Security
  - Cannot build high-security components in Haskell because of the runtime system and heap