Compartmentalized Continuous Integration

David Neto
Senior MTS

Devin Sundaram
Senior MTS

Altera Corp.

THAT SPECIAL THING

2000 That special thing…

2007 p4 vs. svn

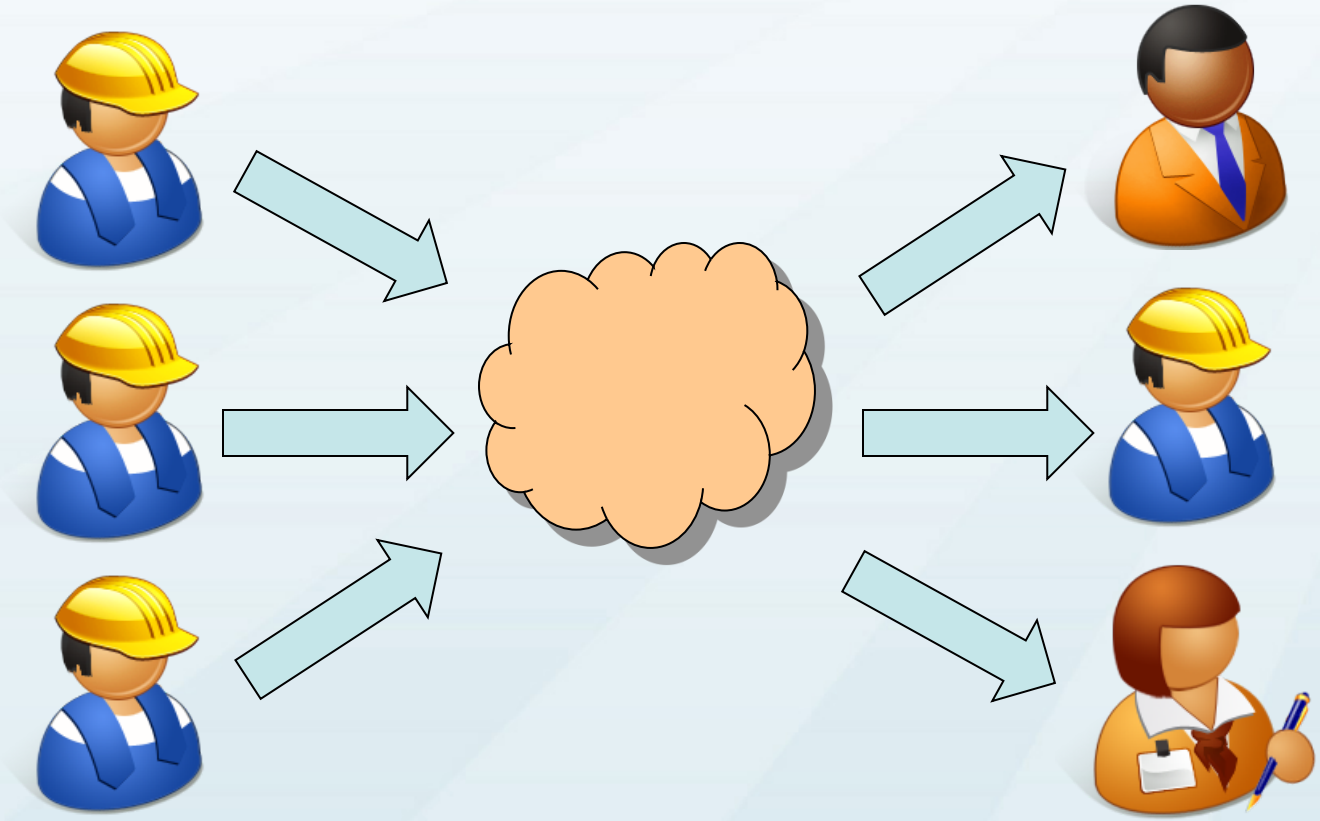
2009 Collaboration++

THREE TAKEAWAYS

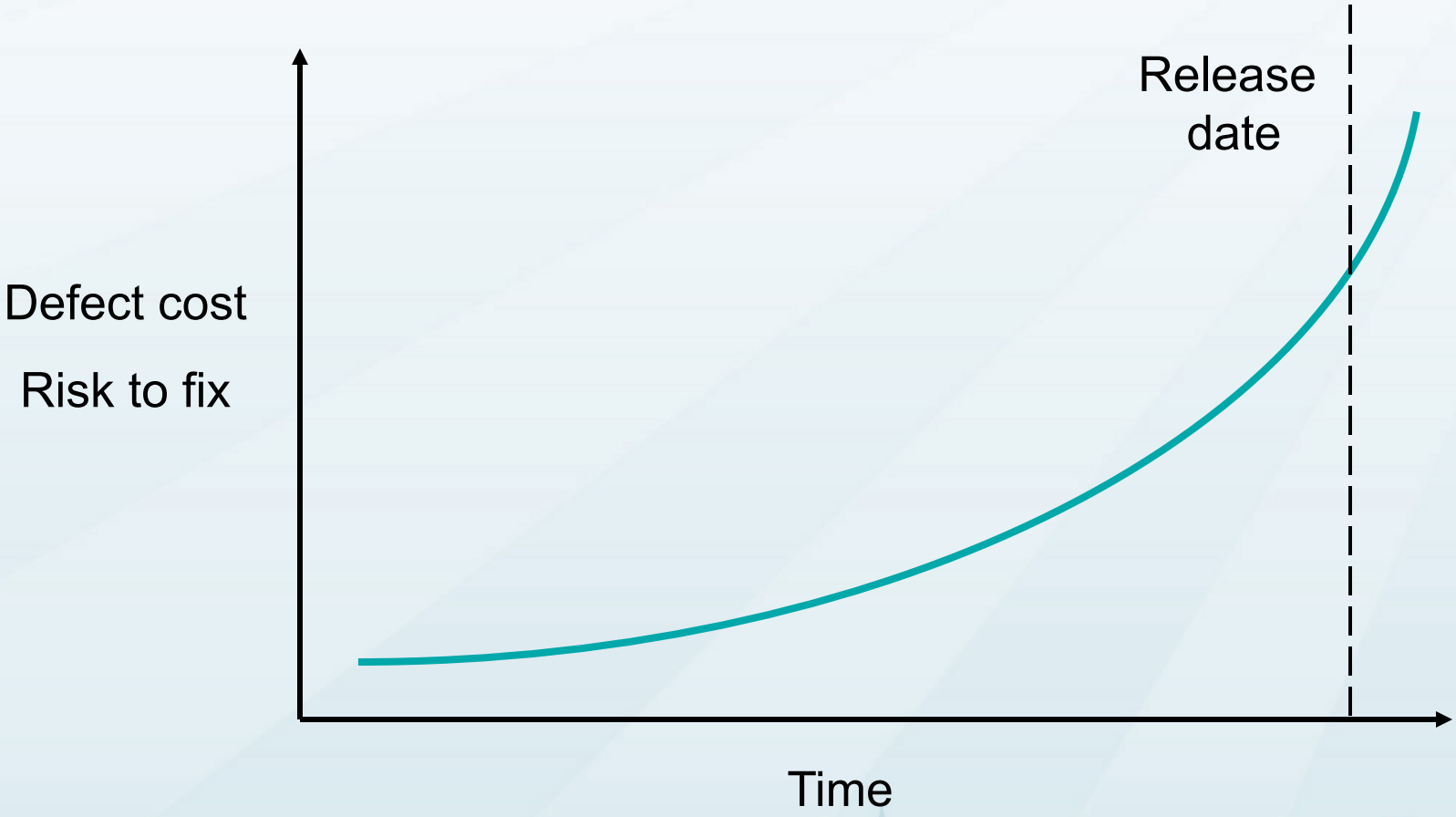
- Continuous Integration is tough with a complex build
- Compartmentalize
  = Classify + filter the change going into your integration build
- Track your own metadata for a codeline
  – With triggers and a second Perforce repository

Continuous Integration

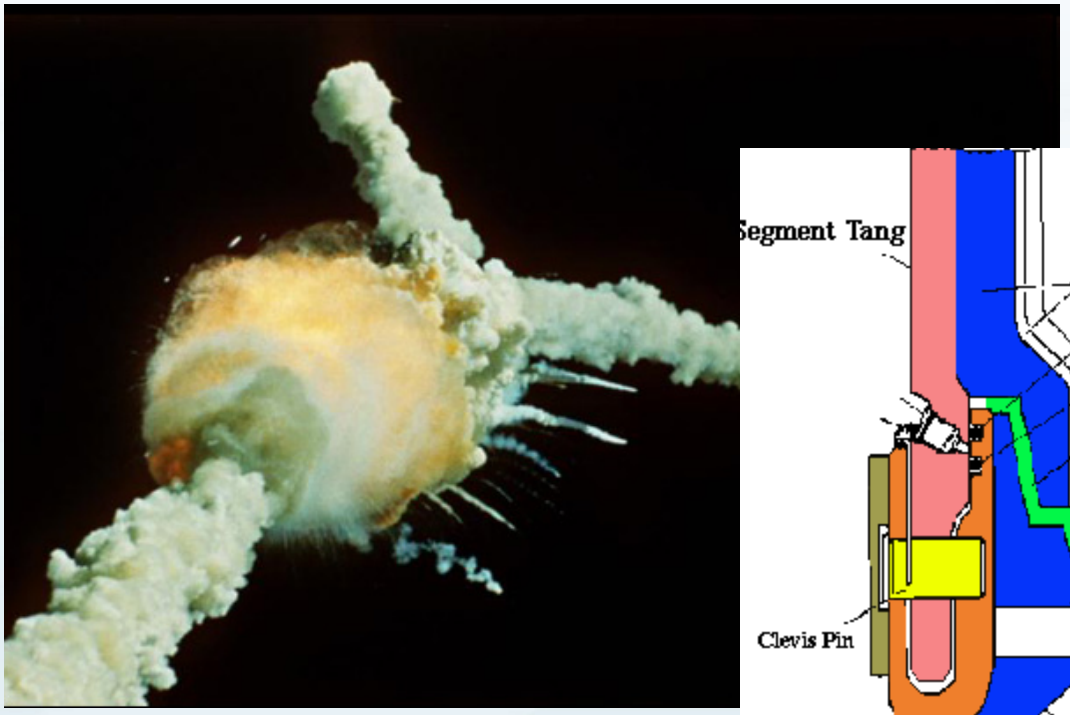
BROADCAST FEATURES / BUGFIX

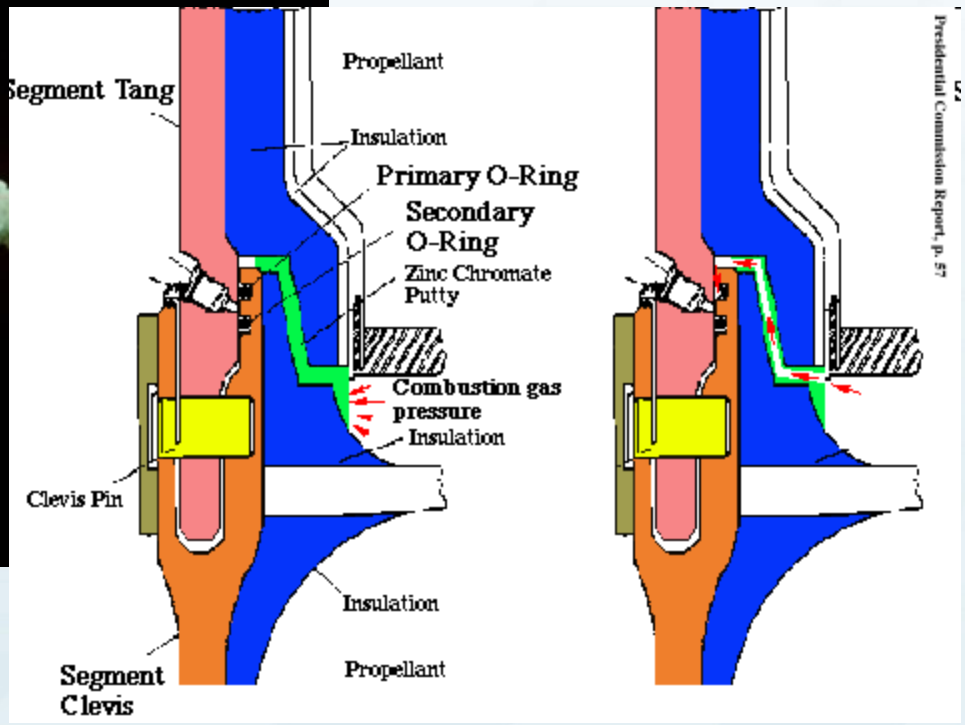


FIND AND FIX DEFECTS *EARLY*



SYSTEMS FAIL AT THE SEAMS





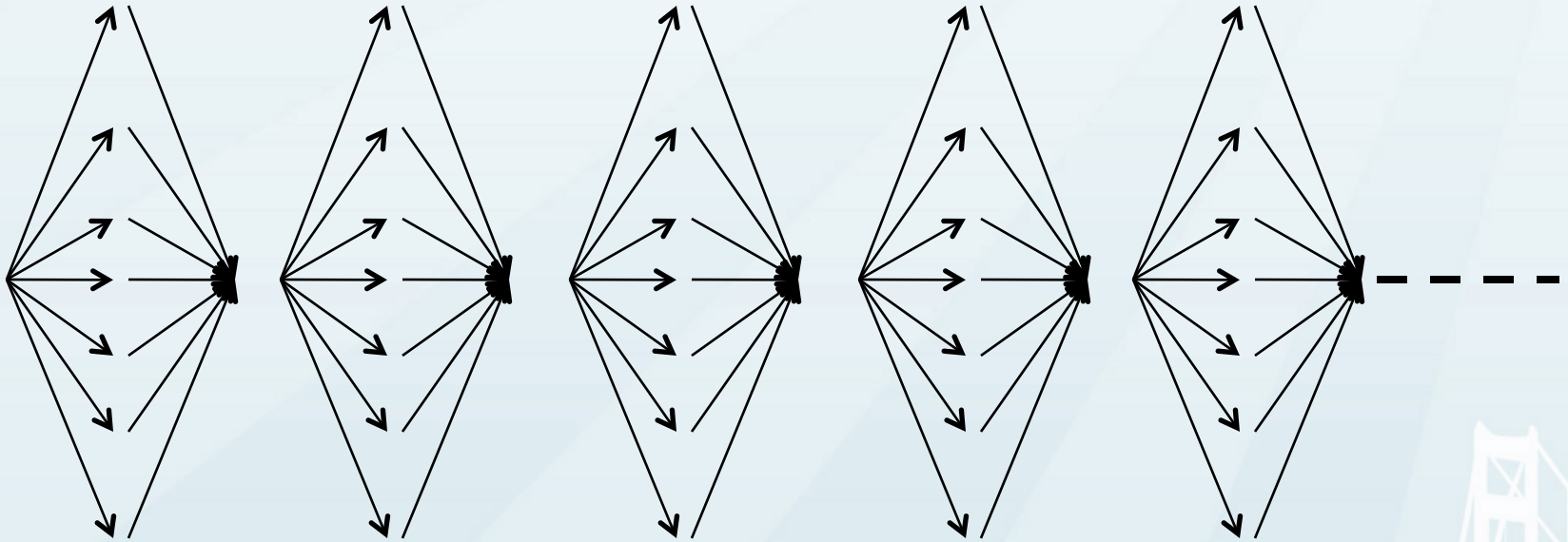
No substitute for end-to-end test

INTEGRATION BUILD *IS* YOUR PRODUCT

- Integration build = put all pieces together
- It's what you deliver. Everything else is just pretend.
- Communicate functionality across your team
  - Broadcast new feature / bugfix
- Complex systems fail at the seams
  - Feedback for developers

CONTINUOUS INTEGRATION

- Make an Integration Build as often as possible
- It's the heartbeat of your project

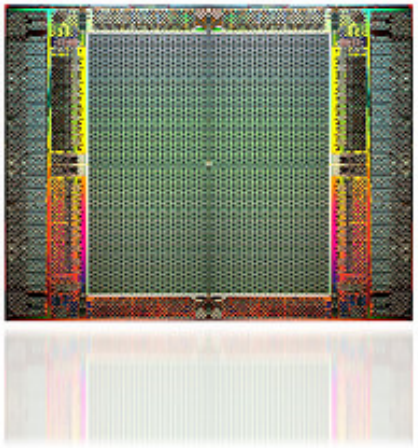


SHAPES ALL PROCESS AND INFRASTUCTURE

- Supporting practices [Fowler]:
  - Maintain a code repository
  - Automate the build
  - Make the build self-testing
  - Commit as often as possible
  - Every commit to mainline should be built
  - **Keep the build fast**
  - Test in a clone of production environment
  - Make it easy to get latest deliverables
  - Everyone can see result of latest build
  - Automate deployment

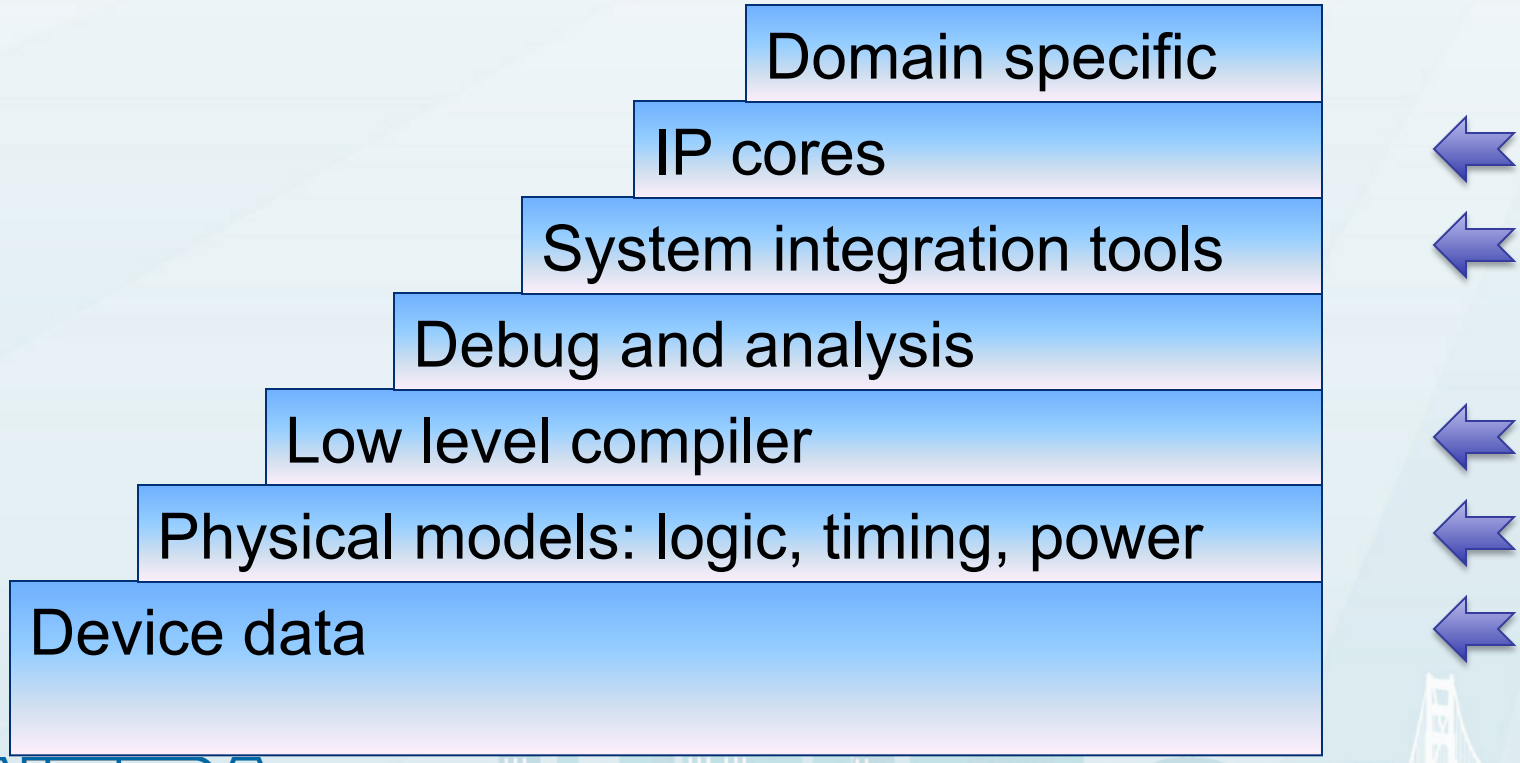
ALTERA'S SOFTWARE BUILD

- Altera makes Field Programmable Gate Arrays (FPGA)
  - Programming = Rewiring
  - 3.9 billion transistors!
- Altera Complete Design Suite (ACDS)
  = Development tools
- ACDS Build:
  - 255K source files, 45GB
  - ~400 developers, 5 locations worldwide
  - **14 hour build**, multiprocessor, multiplatform
  - **Hundreds of source changes per day**



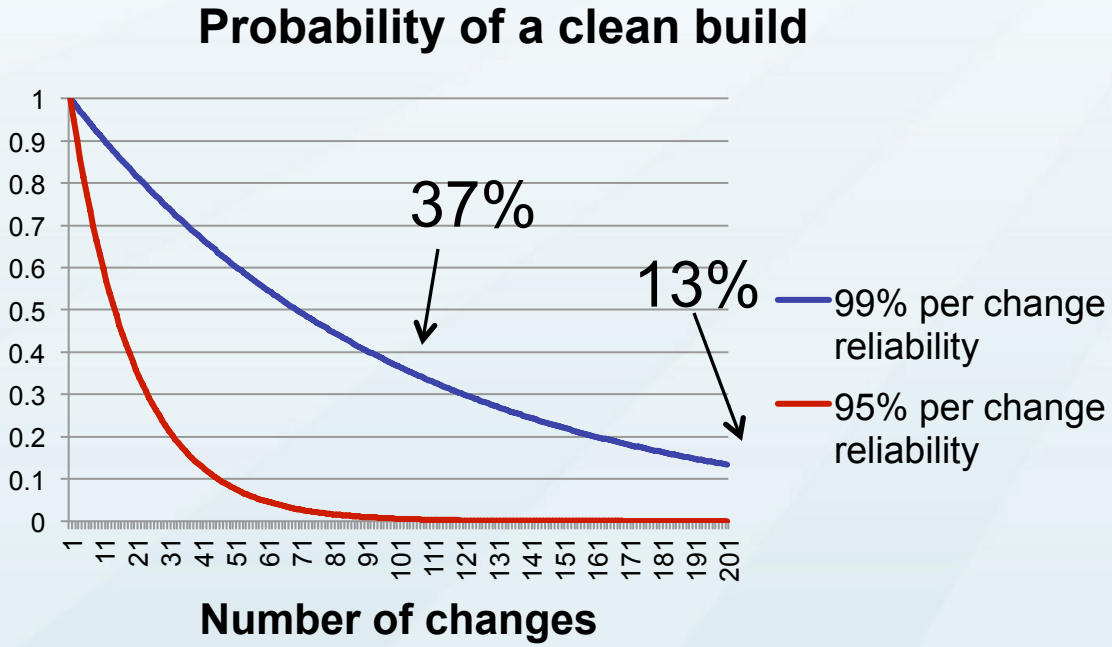
MULTI LAYER SYSTEM CHALLENGE

- Long time to build Device data
- Rapid development within and ***across*** layers
  - E.g. Roll out new device family
  - E.g. DDR memory interface support crosses 5 layers



TOO MANY CHANGES → BUILD RISK

- Probability of a clean build drops quickly with number of new changes



- When it breaks, hard to tell whose fault

STALE BASELINE → COMPOUNDED RISK

- The longer you go without an integration build, the higher the risk
  - Blind to recent data, API, code
- Skip too many heartbeats → ***PROJECT DIES***

SOLUTION: COMPARTMENTALIZATION

- Must keep integration build stable
- Limit the damage to the whole by separating the parts
- But how?

Compartmentalization: Previous approaches

STAGED BUILDS [Fowler]

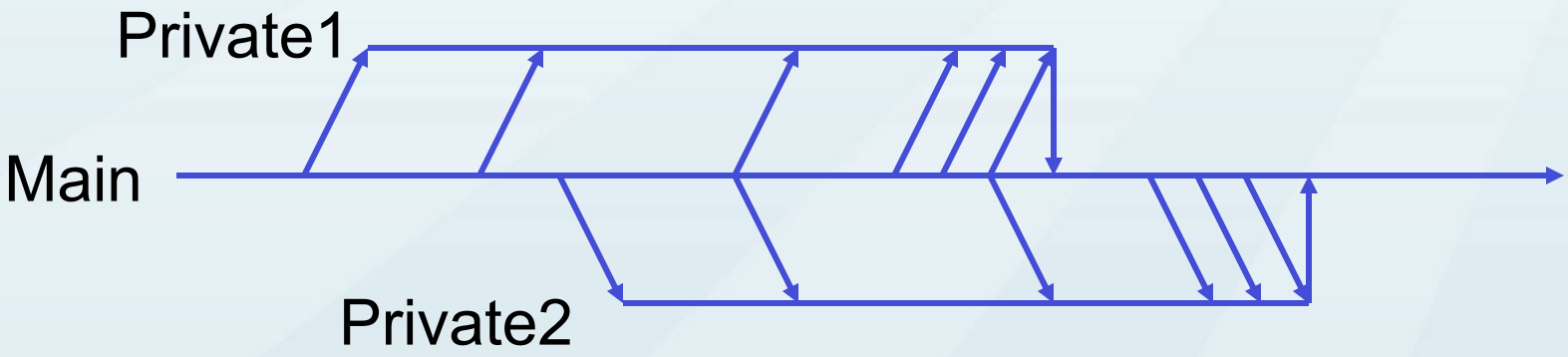
- Full build = pipeline of smaller builds
- Most developers work with output of earlier stage
- In our case: Too slow
  - Device data build = 4 hours
  - Most layers built later: need device info
- **Verdict:** Does not solve our problem

INCREMENTAL REBUILD BOT

- Each change automatically built on latest stable base + changes since stable base
  - Tell developer if it passed or broke the bot
- Tricky policy:
  - If a new change breaks the bot: Keep or Eject?
- In our case:
  - Can't rely on perfect dependencies
  - Device change → full integration build
  - *Apparent* developer reliability improves
- **Verdict:** We use it, but does not solve whole problem

MULTIPLE CODELINES: Strategy

- Partition active work into different codelines
- Qualify separately, with module build
- Frequently integrate “main” → private
- Occasionally integrate private → “main”

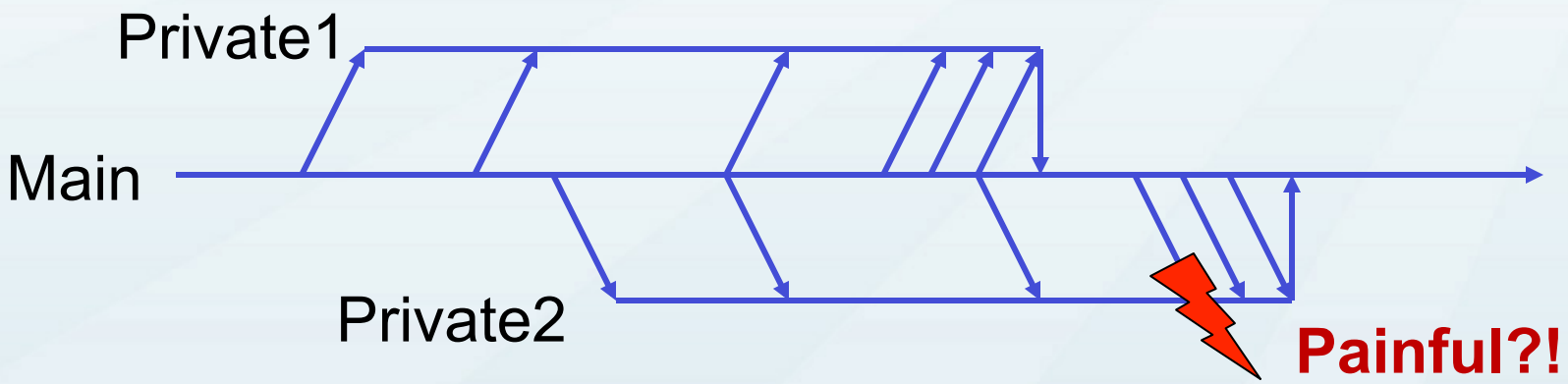


MULTIPLE CODELINES: Variations

- Development codelines [Wingerd]
- Microsoft's Virtual Build Labs [Maraia]
- Inside/Outside codelines, Remote development lines, Change Propagation Queues, … [Appleton et. al.]
- Virtual Codelines (one codeline + just-in-time branching) [Appleton et.al.]

MULTIPLE CODELINES: Issues

- Integration is manual
  - Requires superhero to integrate. Painful.
  - Manual implies infrequent. Delays integration.



- Hard / impossible to develop a change across components

MULTIPLE CODELINES: Verdict

- Ok *if perfect modularity*
- Manual
- Infrequent
- Inflexible: Can't develop across components

- "Occasional" integration, not Continuous!

"*90% of SCM "process" is enforcing codeline promotion to compensate for the* ***lack of a mainline****" -- Wingerd*

Compartmentalization: Altera's solution

REQUIREMENTS

- One codeline
- No client customization: Server side only
- Transparent to most users, most of the time
- Support ad hoc cross-component development
- Automatic: Hands off operation

GATEKEEPER STRATEGY

- *Limit the amount of **untested** code accepted into the integration build*

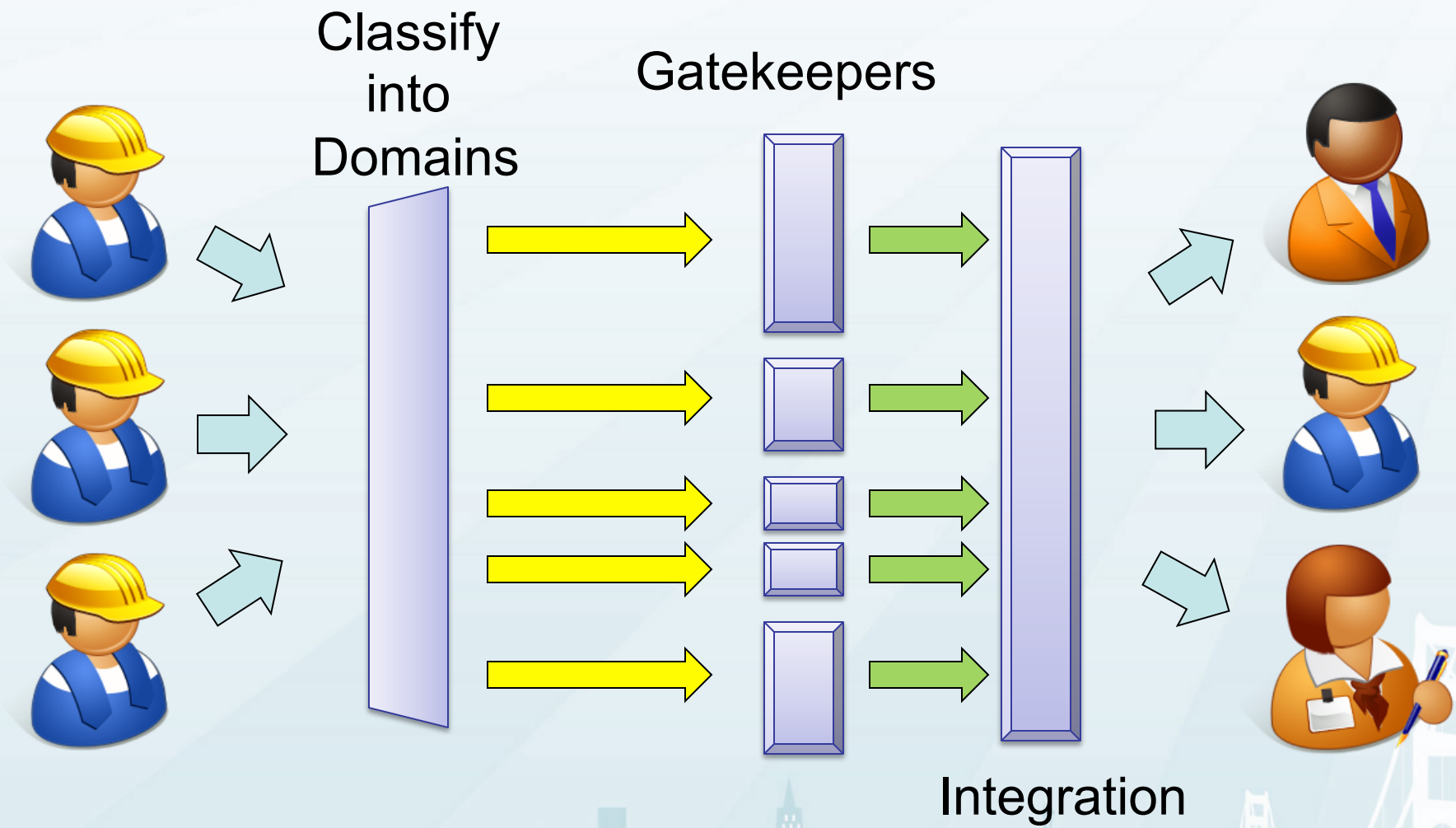
- All code is *guilty* until proven *innocent*
- Integration build uses only *innocent (verified)* code*

- Each file revision in one of two **integration states**:

- Upgrade from **Fresh** to **Verified** when used in a successful **Gatekeeper** build

*Some exceptions

COMPARTMENTALIZE = CLASSIFY + FILTER



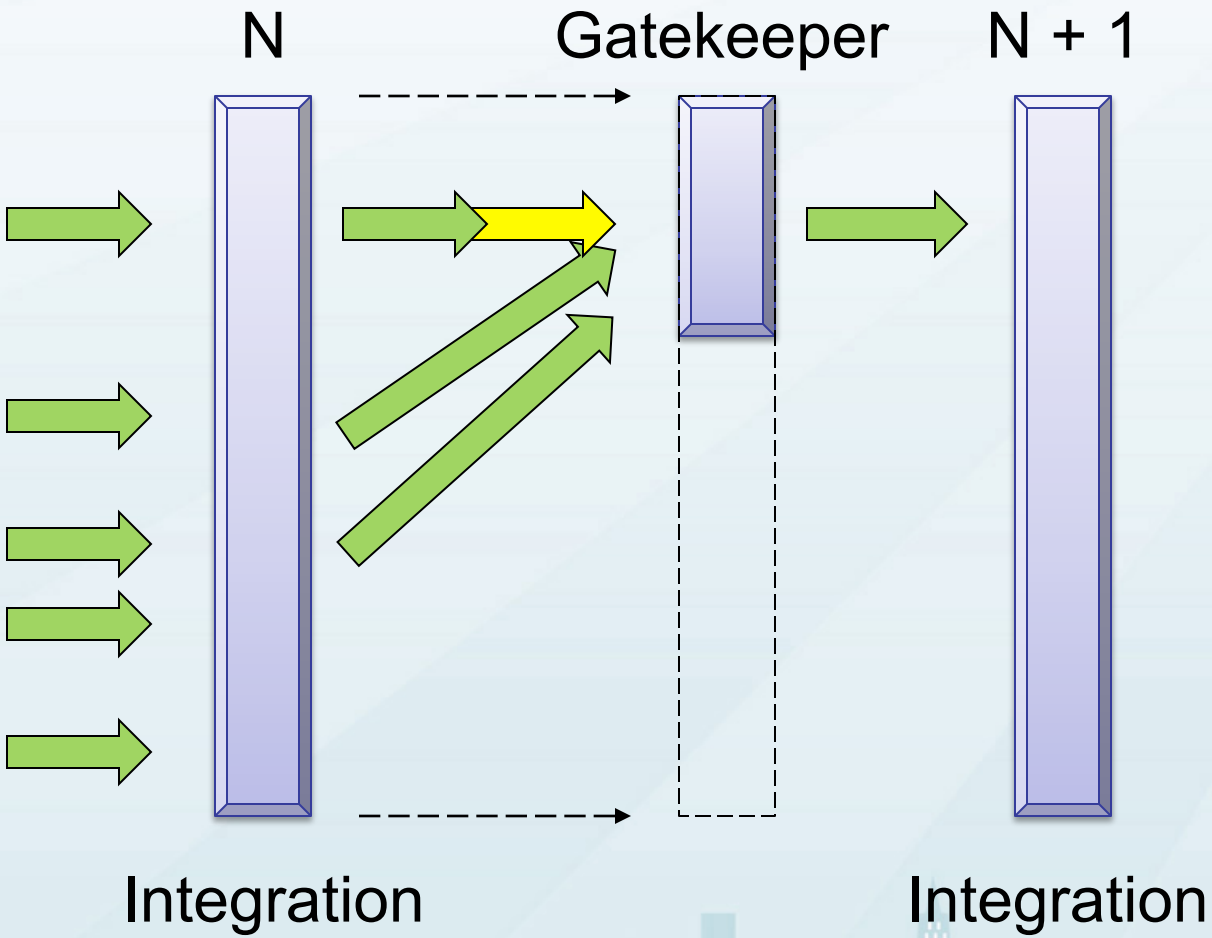
CLASSIFICATION: ZONES, DOMAINS

- Classify each submitted change into a **domain**
- **Site** = Dev location that makes an integration build
- **Zone** = named set of depot paths
  - One zone for each major component
  - Zone can be “site specific”
    - When lots of activity in that zone, and want to protect a site from bad changes from other sites
- **Domains** =
  - Zone
  - { Zone:Site | for each Site, each site-specific Zone }
  - COMBO
    - If a change touches files in more than one Zone

GATEKEEPER RESPONSIBILITY

- Each Gatekeeper is **responsible** for a Domain
  - Validates Fresh changes in that Domain
- Run part or all of the build
  - Uses Fresh revisions from its own Domain
  - Verified code otherwise
- If ok, update integration state:

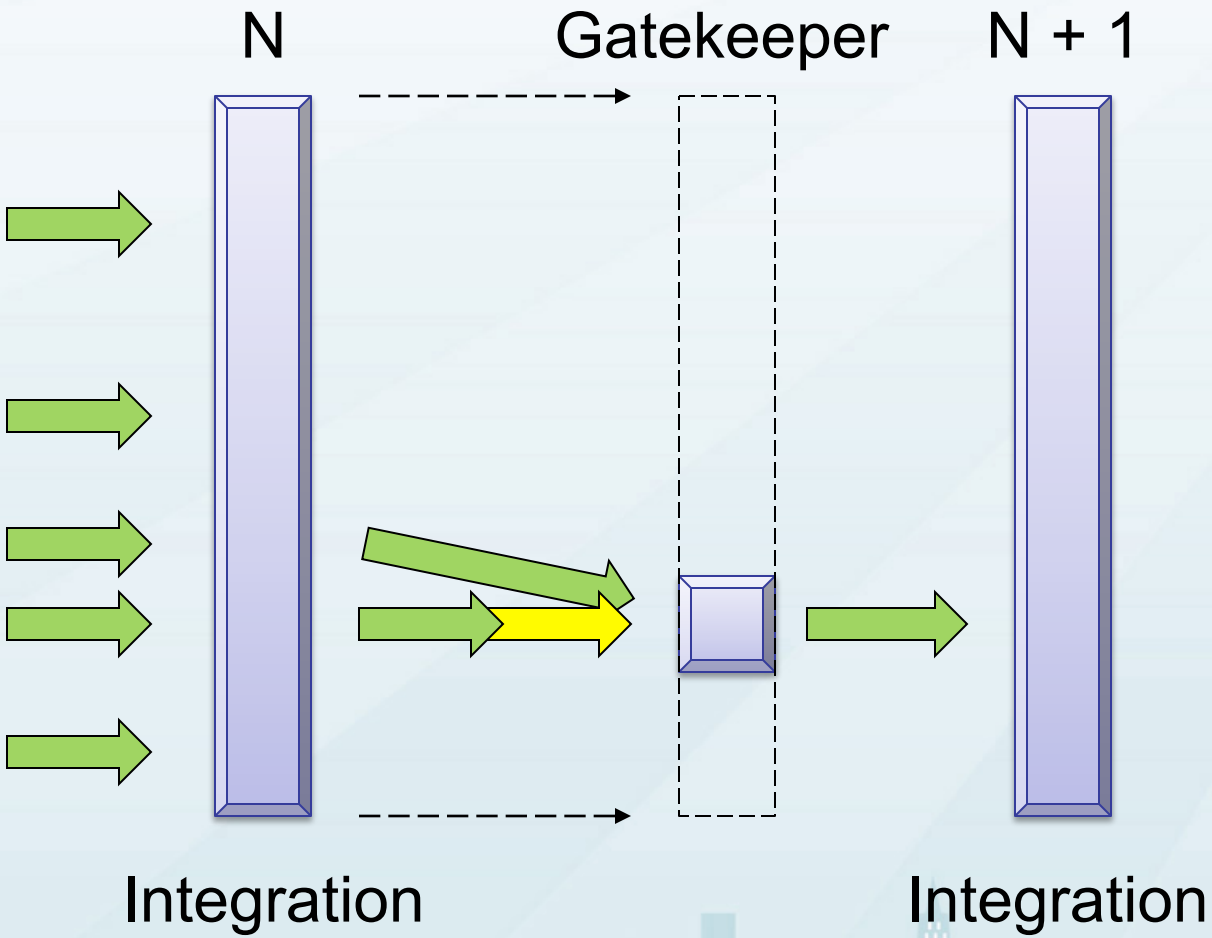
EXAMPLE GATEKEEPER



Runs *part* of the build, on top of previous full build.

Responsible for one domain, uses verified source from two others.

OTHER GATEKEEPER: *SPREAD + LIMIT RISK*



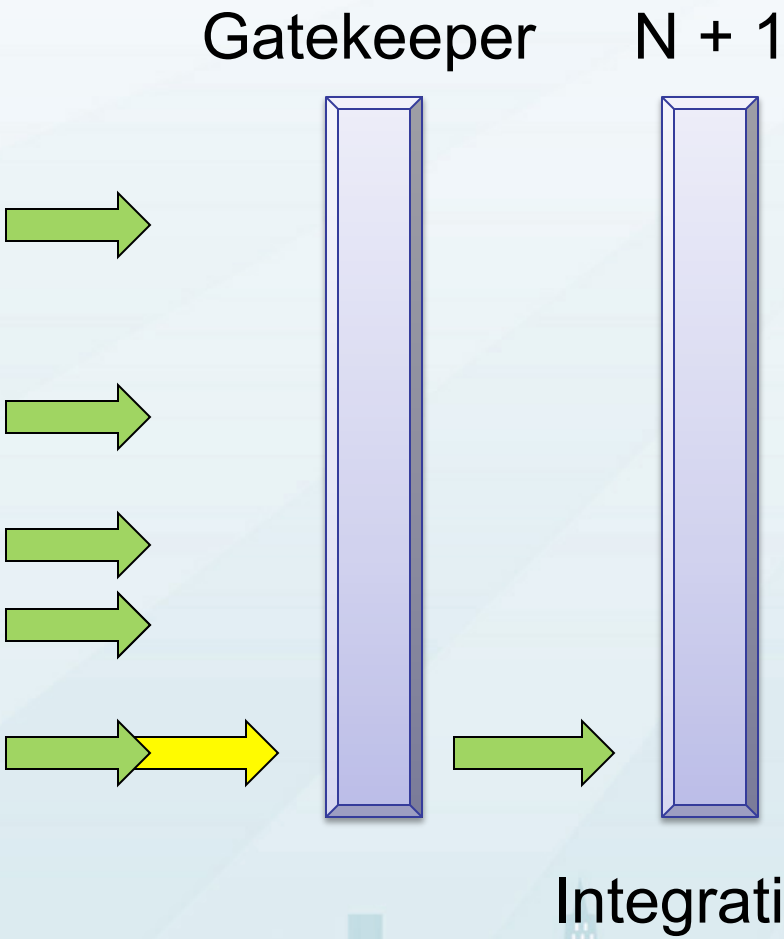
In general, limited amount of

Fresh

change going into any one build.

Climb the reliability curve!

GATEKEEPER CAN RUN WHOLE BUILD



But responsible for just one domain.

COMBO builds do this

EXCLUSION RULE

- Should avoid “broken by construction” gatekeepers
- Rule: ***Each file may have fresh revisions from at most one domain***
  - Conflicts from: Site-specific zones; COMBO
- Allow many fresh revisions from same domain
  - Enable rapid development

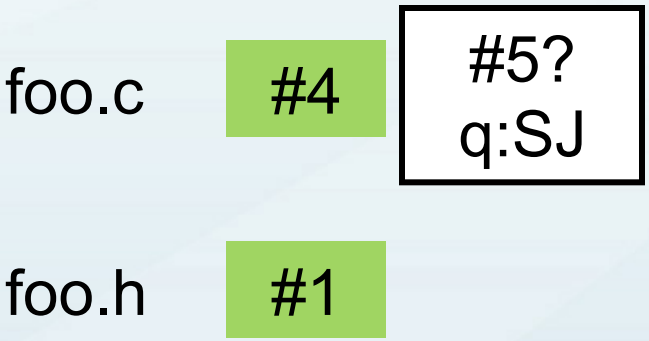
E.g. Alice (site TO) submits foo.c, foo.h

	Alice changed param type		q:SJ Gatekeeper uses	q:TO Gatekeeper uses
foo.c	#4	#5 q:TO	#4	#5 q:TO
foo.h	#1	#2 q:TO	#1	#2 q:TO

Zone “q” is site-specific
TO, SJ are sites

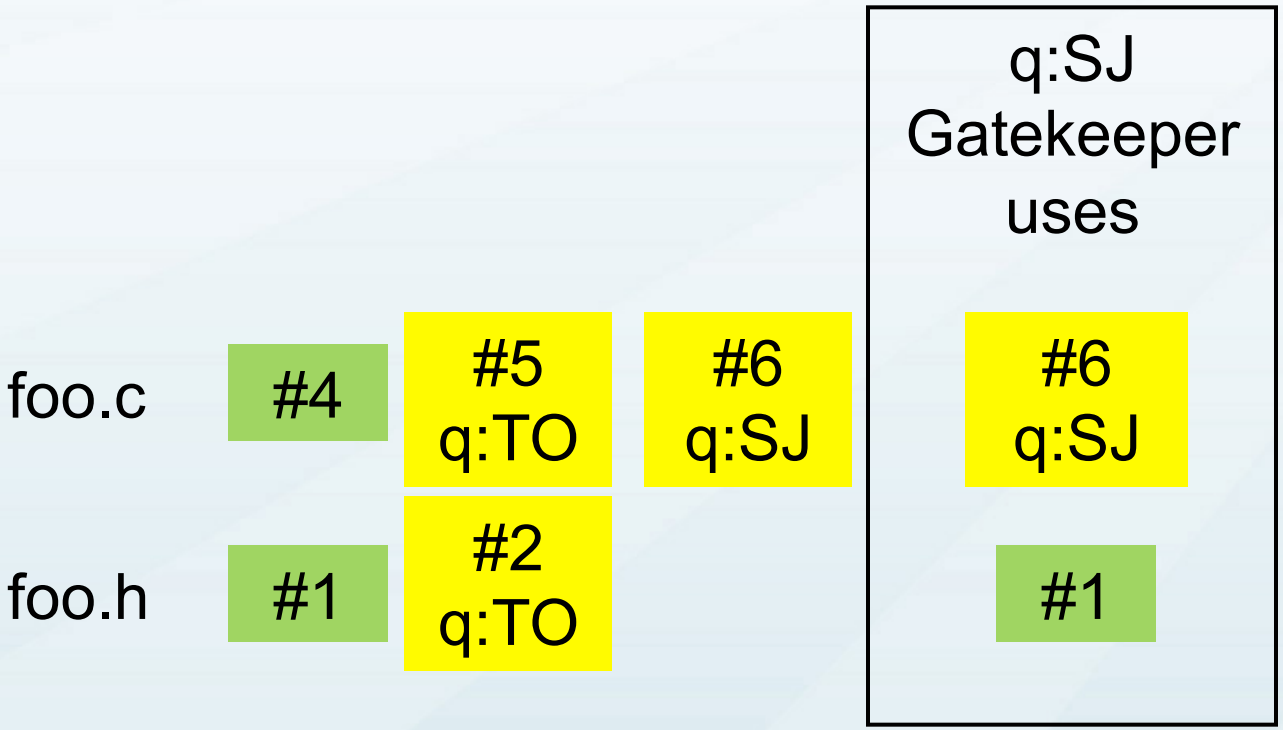
Bob (site SJ) develops update to foo.c …

Bob does not know about Alice's change



Bob resolves to Alice's change

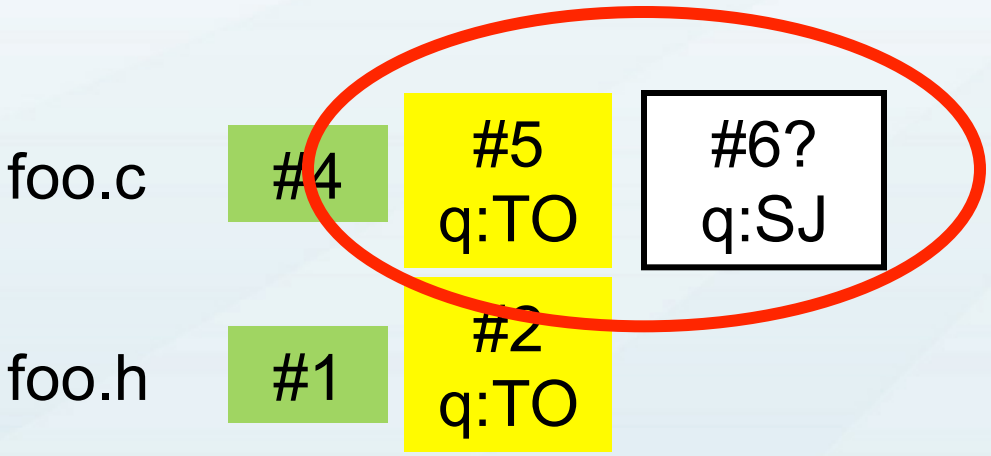
What if we allow Bob to submit?



Sees only half of Alice's change!

BROKEN BY CONSTRUCTION

Exclusion Rule avoids broken-by-construction



Exclusion rule detects this conflict,
Rejects Bob's change

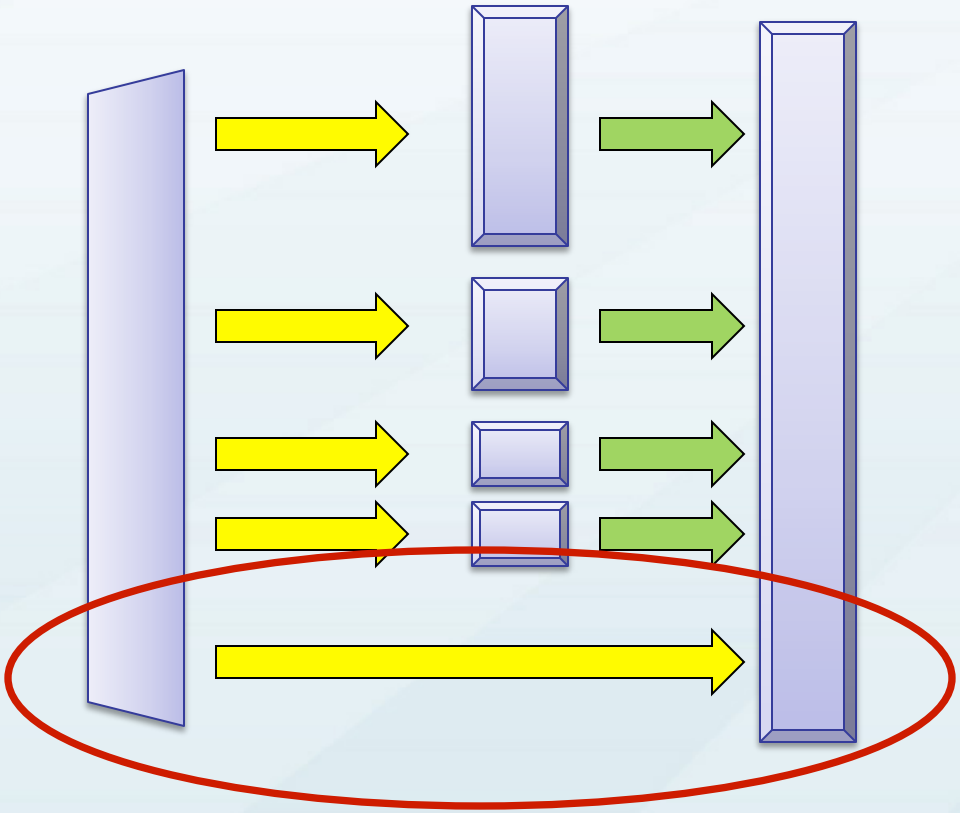
Bob waits until Alice's change is verified

Now Bob's change is accepted

NOMADIC OWNERSHIP

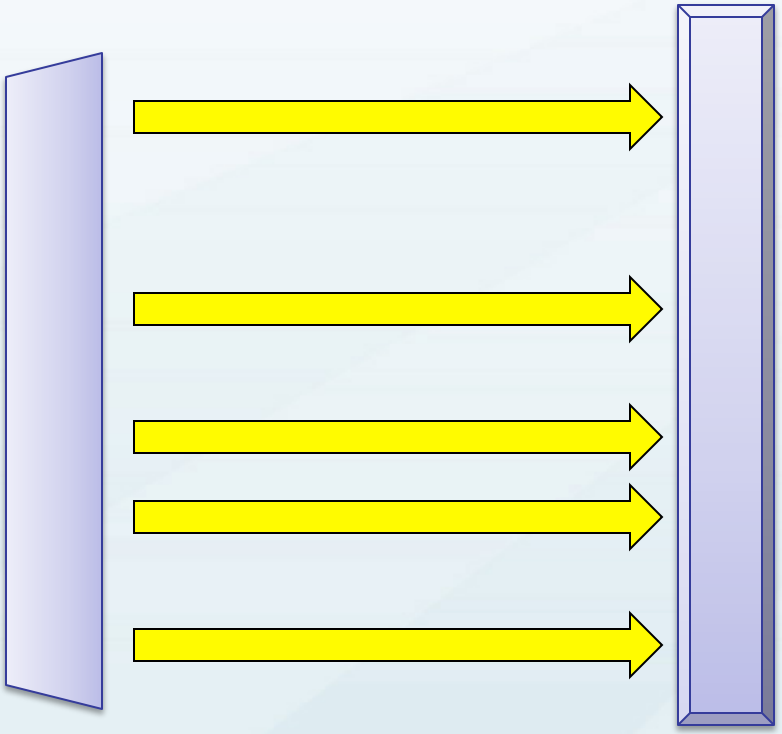
- Exclusion Rule creates *temporary* “ownership” of a file
  - Delays updates destined to other domains
  - Especially within site-specific zones
- Sometimes annoying
  - Willing to pay the price
  - Better than the alternatives!
- Minimized by refactoring: break up files
- But it’s flexible and automatic
  - Temporary ownership migrates according to update patterns

SOMETIMES BYPASS GATEKEEPERS



- Rare, long build time
  - E.g. COMBO
- Site-protected, long build time
- Acceptable integration risk

TURN OFF GATEKEEPERS



- Late in release cycle
  - Development has slowed
  - Each change carefully reviewed
- Low integration risk
- Avoid annoyance of Exclusion Rule

Mechanics

INTEGRATION STATUS TRACKING: WHAT

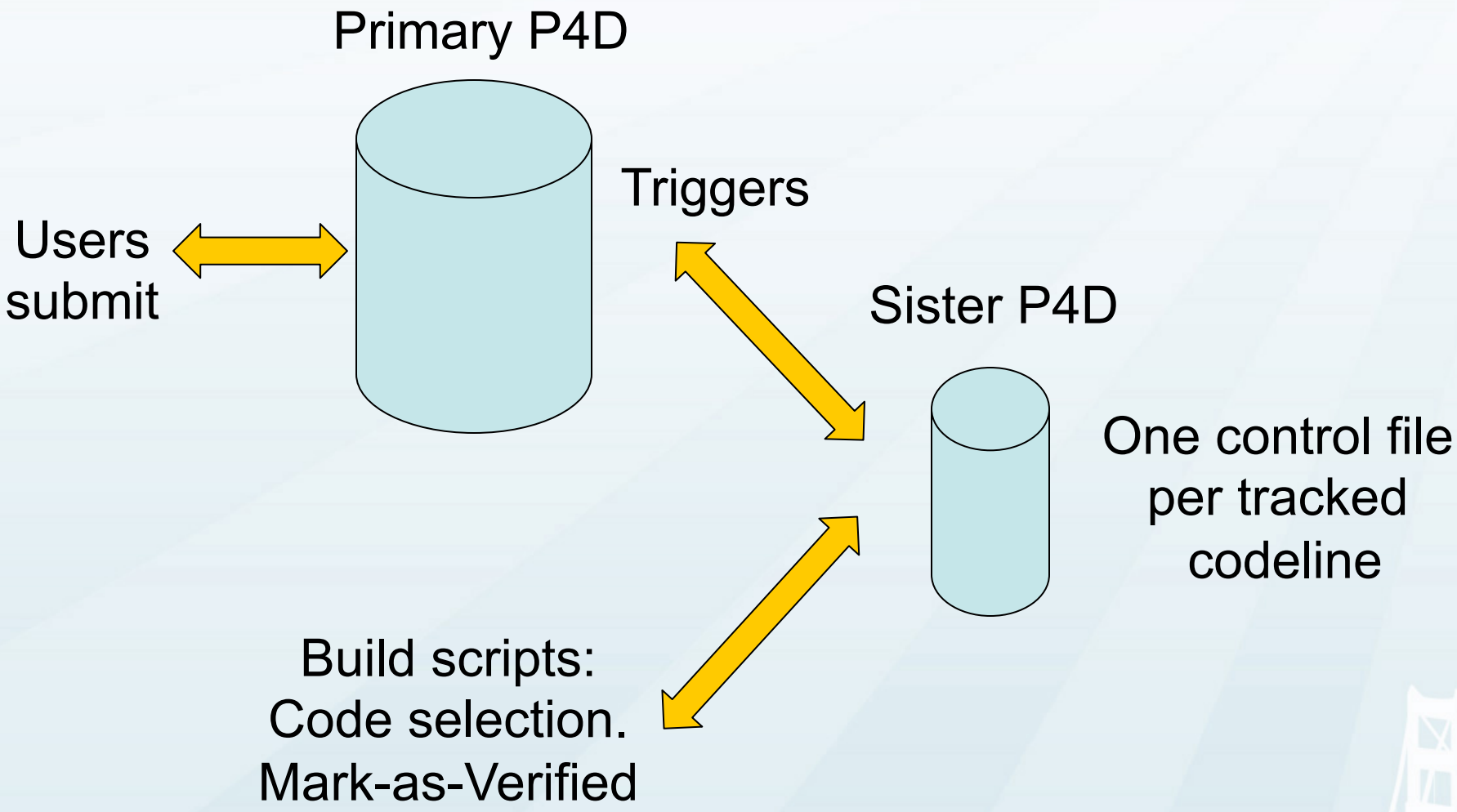
- For each file revision keep:
  - State: Fresh, or Verified
  - Domain
  - User, change#, depot path
- Store in log-structured **control file**
- But only need this for *recent* revisions
  - Only Fresh revisions can conflict
  - Each revision eventually Verified
  - Need only from oldest Fresh until #head
    - Purge older records

INTEGRATION STATUS TRACKING: HOW

- Integration status must always be up-to-date
  - Needed for Exclusion Rule, checked in change-submit trigger
- Can't just use floating labels:
  - P4 triggers can't update P4 metadata !

- Need:
  - Fast atomic updates to control file
  - Only *need* latest version
  - Compact storage
- Store in a second Perforce repository!
  - Filetype text+S512: Purges old contents

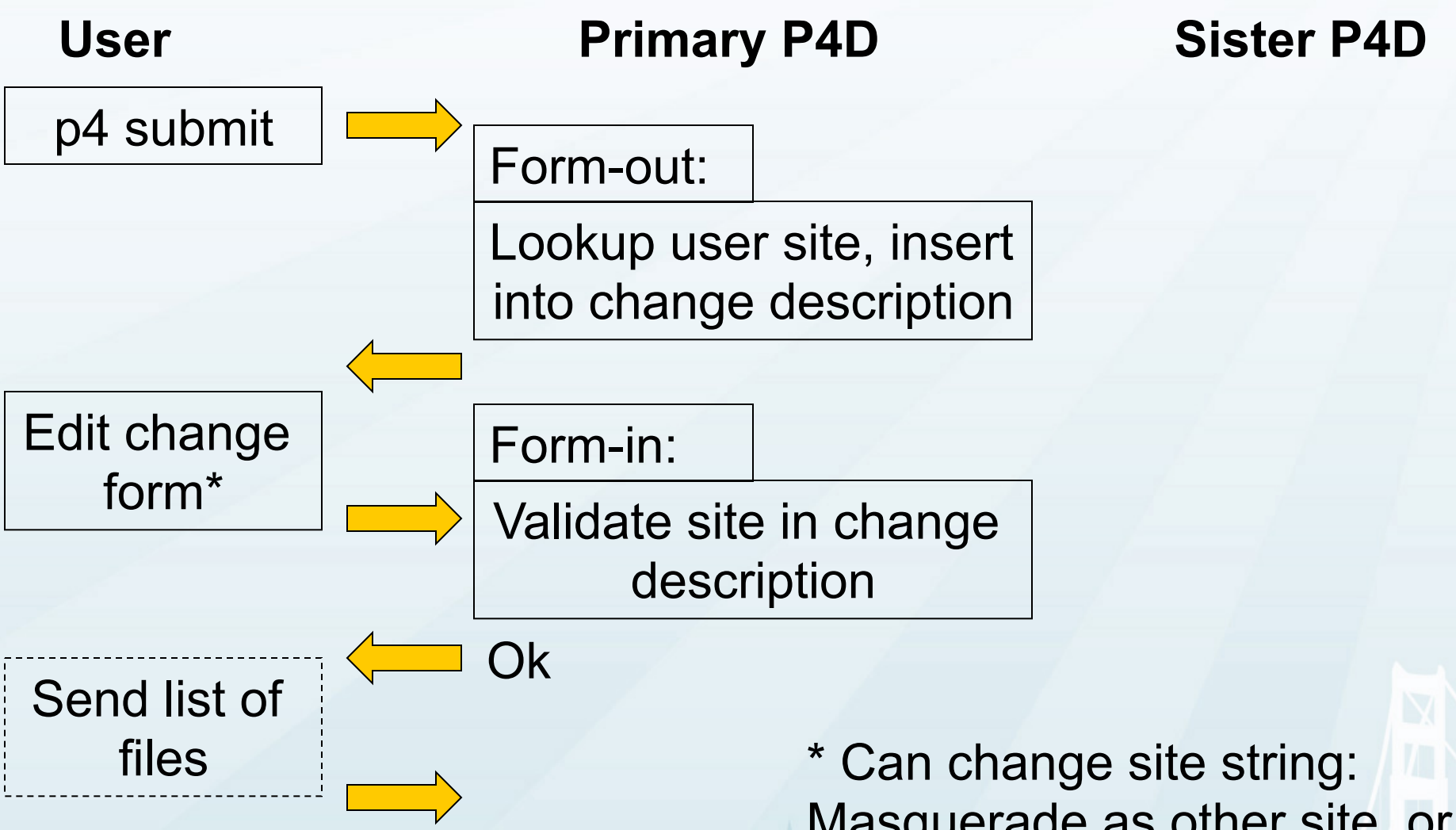
USING A SECOND PERFORCE REPOSITORY



CONFIGURATION

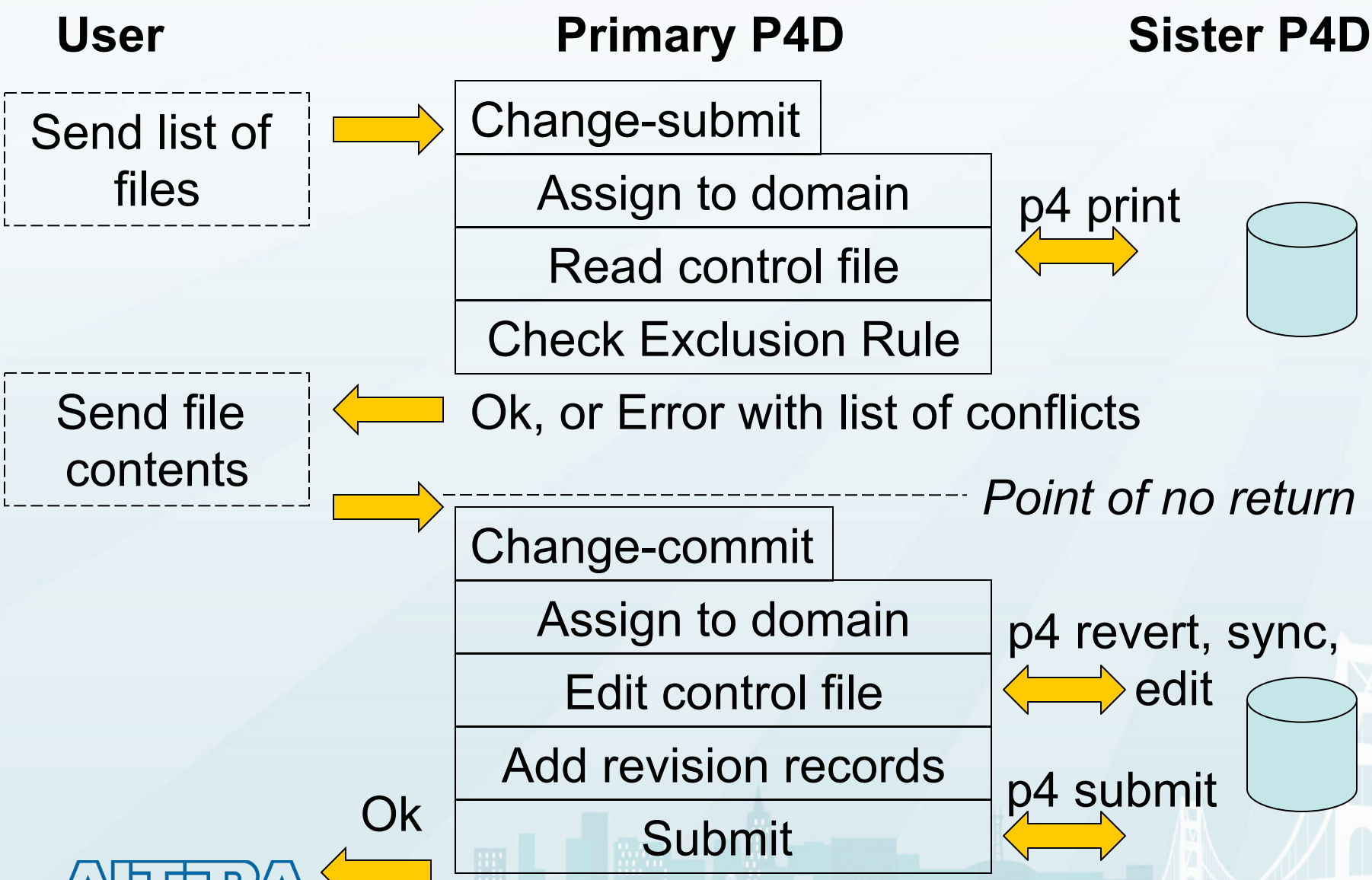
- Map developers to sites via P4 Group membership
  - E.g. p4sip.to.users, p4sip.sj.users
- Codeline policy defined in “p4sip.zone” file
  - Stored in root of codeline: Carried into branches
  - Sites
  - Named zones
    - Mapping to depot paths relative to the codeline
    - Which zones are site-specific
  - Parse output of “p4 print”: Safe in triggers
- Triggers
  - From-out, form-in: “change” forms
  - Per codeline: change-submit, change-commit

LIFE CYCLE OF A CHANGE SUBMISSION (1/2)



* Can change site string: Masquerade as other site, or force COMBO

LIFE CYCLE OF A CHANGE SUBMISSION (2/2)



MAKING A BUILD

1. Produce “build label”: revisions to use
   - Select: Base build label + base domains, Verified domains, Fresh domains
   - Use “painter algorithm”:
     - For each file, take latest revision in any category
2. Compile + smoketest (subset of) code
3. If not ok:
   - Notify errant developer
   - If integration build: Repair build, update label
4. If ok:
   - Publish build label along with binary
   - Mark revisions in label as Verified
     - Use same revert/sync/edit/submit logic as the commit trigger

Summary

WHY DOES IT WORK?

- ***Climb the reliability curve***

- Gatekeepers catch most breaks
- Run Gatekeepers often: usually fast
- Reduce “fog of war” in Integration build

USABILITY

- Most people never notice
- Exclusion rule can annoy
- “Build temporal mechanics”
  - Uncertain delay between submit and getting into the build
  - Changes appear out of order in different sites

EFFECTIVENESS

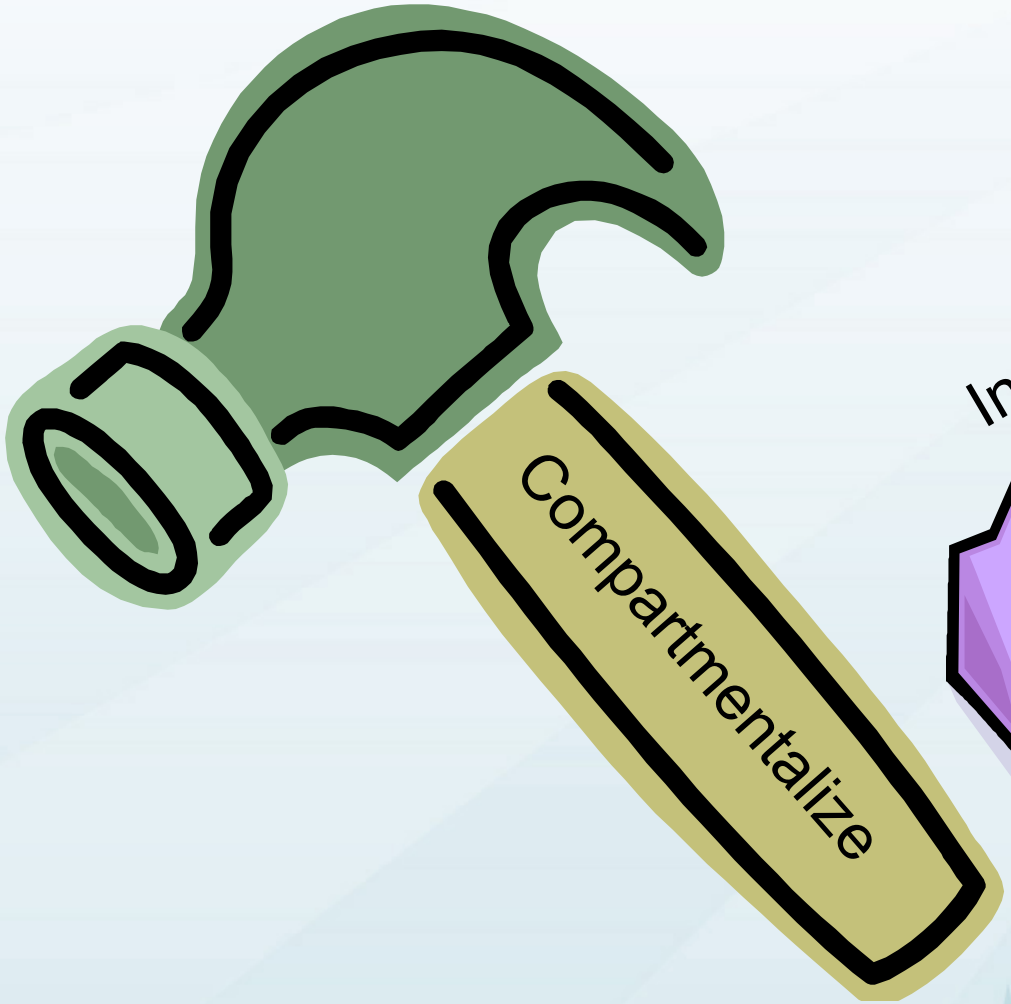
- Integration build each day x 3 sites
- Changes from site X
  - In site X's integration build *next day*
  - In site Y's integration build 36-48 hours later

- No performance loss
- Control file typically ~ 2K revision records
- 175,000 changes in 3 years

OTHER USES FOR YOUR OWN METADATA?

- Can track whatever metadata you want
  - Within a “recent” time window
- Change review and approval
- Nested transactions?
  - “Group commit” emerges from Exclusion rule
    - Alice, Andrew, Amy commit changes in TO
    - SJ build sees all or none of them
  - Could generalize this…?

ANOTHER TOOL FOR YOUR TOOLBOX



Acknowledgements

- Rob Romano
- Jeff Da Silva
- David Karchmer
- Mun Soon Yap
- Addy Yeow
- Alan Herrmann

- And all the developers and builders at Altera

Thank You!