Designing CI-Flows

CI for thousands of developer

November 2, 2020

Disclaimer:

- Layman's experience from the trenches
- AKA: *Davids opinion considered harmful!*

Disclaimer:

- Layman's experience from the trenches
- AKA: *Davids opinion considered harmful!*

Disclaimer:

- Layman's experience from the trenches
- AKA: *Davids opinion considered harmful!*

Who am I

- Sigma and Ericsson since 2014

Who am I

- Sigma and Ericsson since 2014
  - Radio Base Station

Who am I

- Sigma and Ericsson since 2014
  - Radio Base Station
- Feature developer $\rightarrow$ troubleshooter $\rightarrow$ development environment

Who am I

- Sigma and Ericsson since 2014
  - Radio Base Station
- Feature developer $\rightarrow$ troubleshooter $\rightarrow$ development environment$\rightarrow$ Flow Guardian

Who am I

- Sigma and Ericsson since 2014
  - Radio Base Station
- Feature developer → troubleshooter → development environment→ Flow Guardian

Ericsson RBS

Ericsson RBS

- Thousands of developers
- All developing for Radio Base Station
  - Different sub-organizations, different responsibilities
  - Sigma, one sub-org $\approx$ 700 developers
- Gerrit / Git / Jenkins / Jira / (Eiffel)
- + in-house tools

Ericsson RBS

- Thousands of developers
- All developing for Radio Base Station
  - Different sub-organizations, different responsibilities
  - Sigma, one sub-org $\approx$ 700 developers
- Gerrit / Git / Jenkins / Jira / (Eiffel)
- + in-house tools

Ericsson RBS

- Thousands of developers
- All developing for Radio Base Station
  - Different sub-organizations, different responsibilities
  - Sigma, one sub-org $\approx$ 700 developers
- Gerrit / Git / Jenkins / Jira / (Eiffel)
- + in-house tools

Ericsson RBS

- Thousands of developers
- All developing for Radio Base Station
  - Different sub-organizations, different responsibilities
  - Sigma, one sub-org $\approx$ 700 developers
- Gerrit / Git / Jenkins / Jira / (Eiffel)
- + in-house tools

Ericsson RBS

- Thousands of developers
- All developing for Radio Base Station
  - Different sub-organizations, different responsibilities
  - Sigma, one sub-org $\approx$ 700 developers
- Gerrit / Git / Jenkins / Jira / (Eiffel)
- + in-house tools

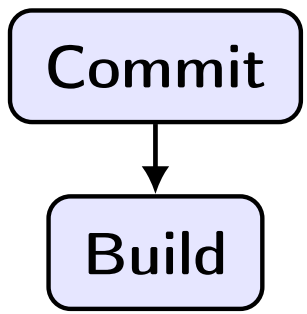
Ericsson RBS

- Thousands of developers
- All developing for Radio Base Station
  - Different sub-organizations, different responsibilities
  - Sigma, one sub-org $\approx$ 700 developers
- Gerrit / Git / Jenkins / Jira / (Eiffel)
- + in-house tools

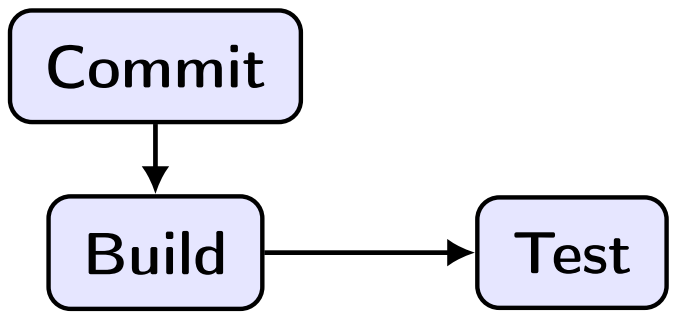
What is CI for you?

Basic CI

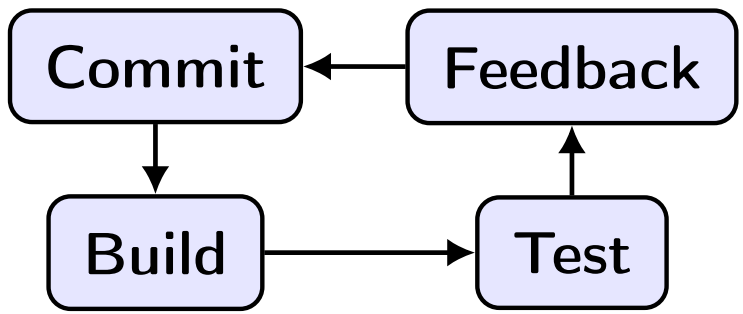
Basic CI



Basic CI



Basic CI



Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

CI from the CI Flow Plumber's point of view

- Modularization
- Logging

- Non-exhaustive list!
  - Speed
  - Stability
  - Scalability
  - Reproducibility
  - ...

CI from the CI Flow Plumber's point of view

- Modularization
- Logging

- Non-exhaustive list!
  - Speed
  - Stability
  - Scalability
  - Reproducibility
  - ...

CI from the CI Flow Plumber's point of view

- Modularization
- Logging

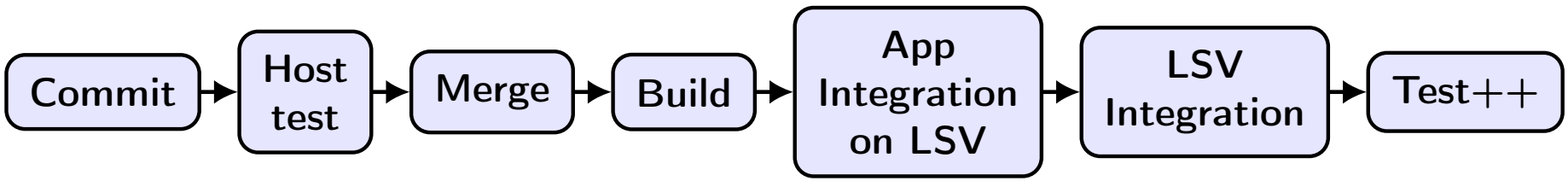
- Non-exhaustive list!
  - Speed
  - Stability
  - Scalability
  - Reproducibility
  - ...

CI from the CI Flow Plumber's point of view

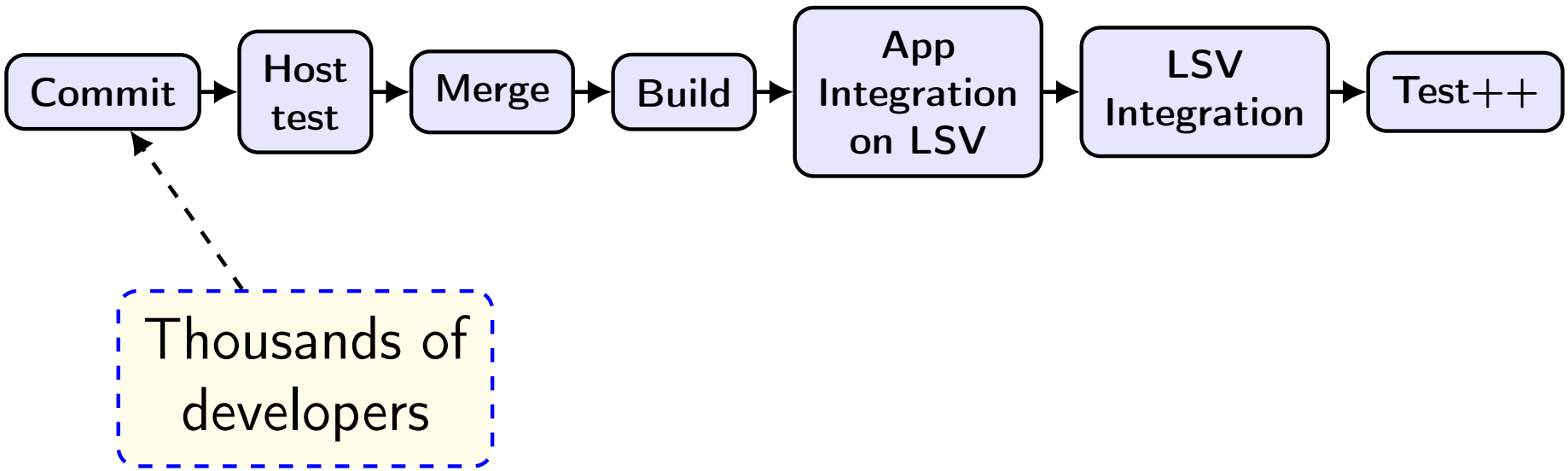
- Modularization
- Logging

- Non-exhaustive list!
  - Speed
  - Stability
  - Scalability
  - Reproducibility
  - ...

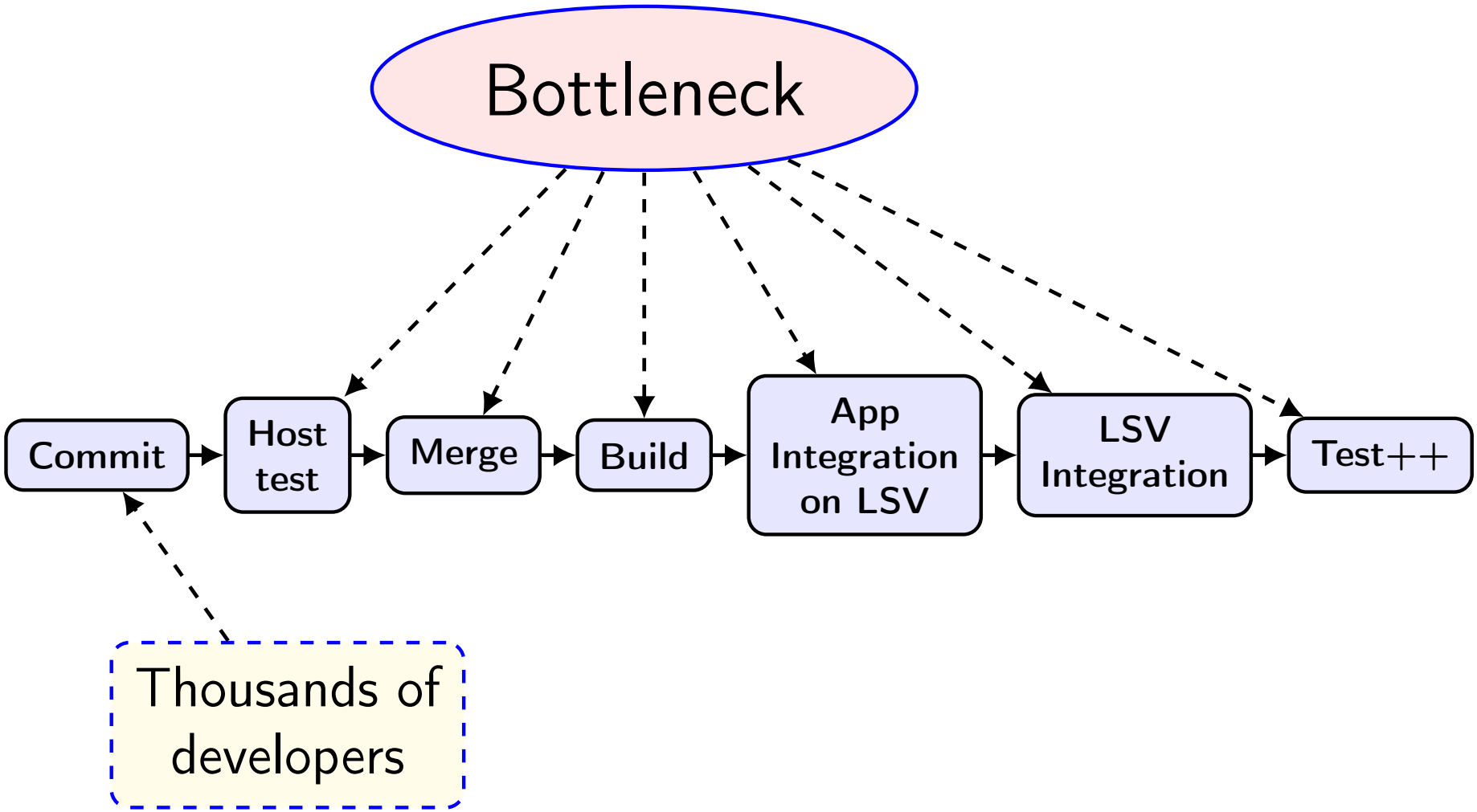
Modularization



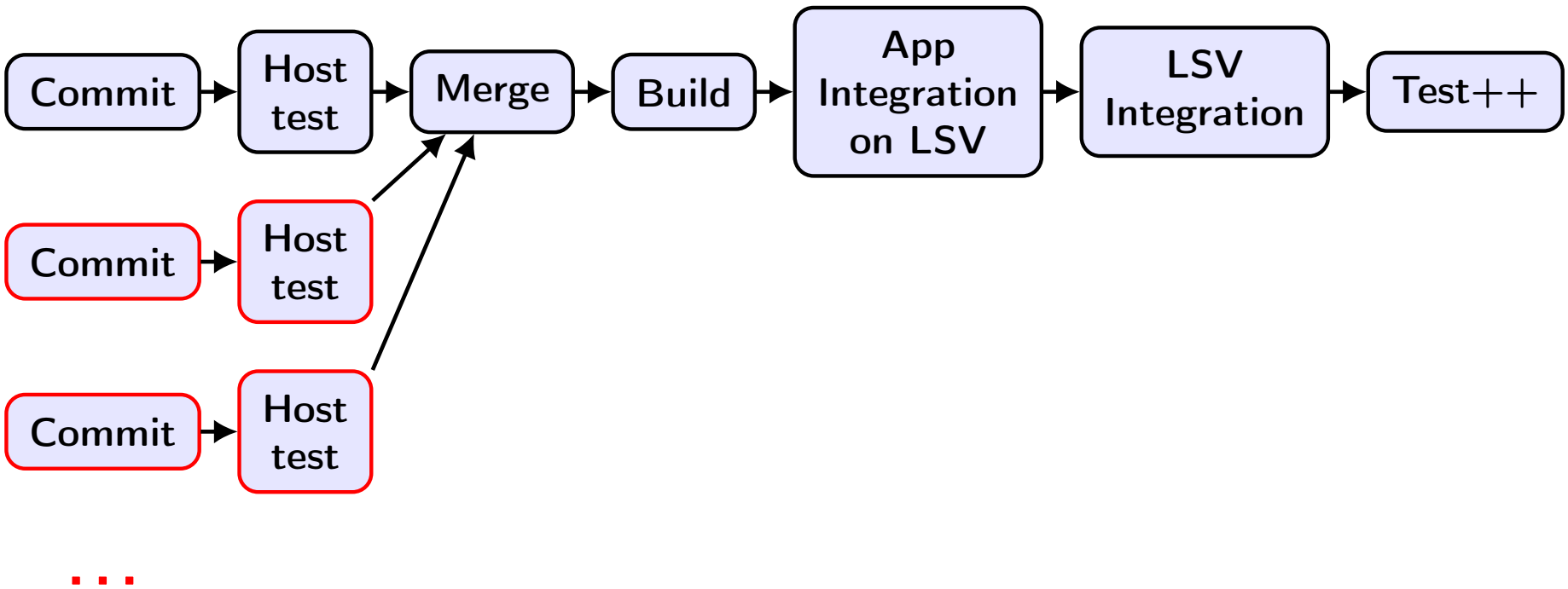
Modularization



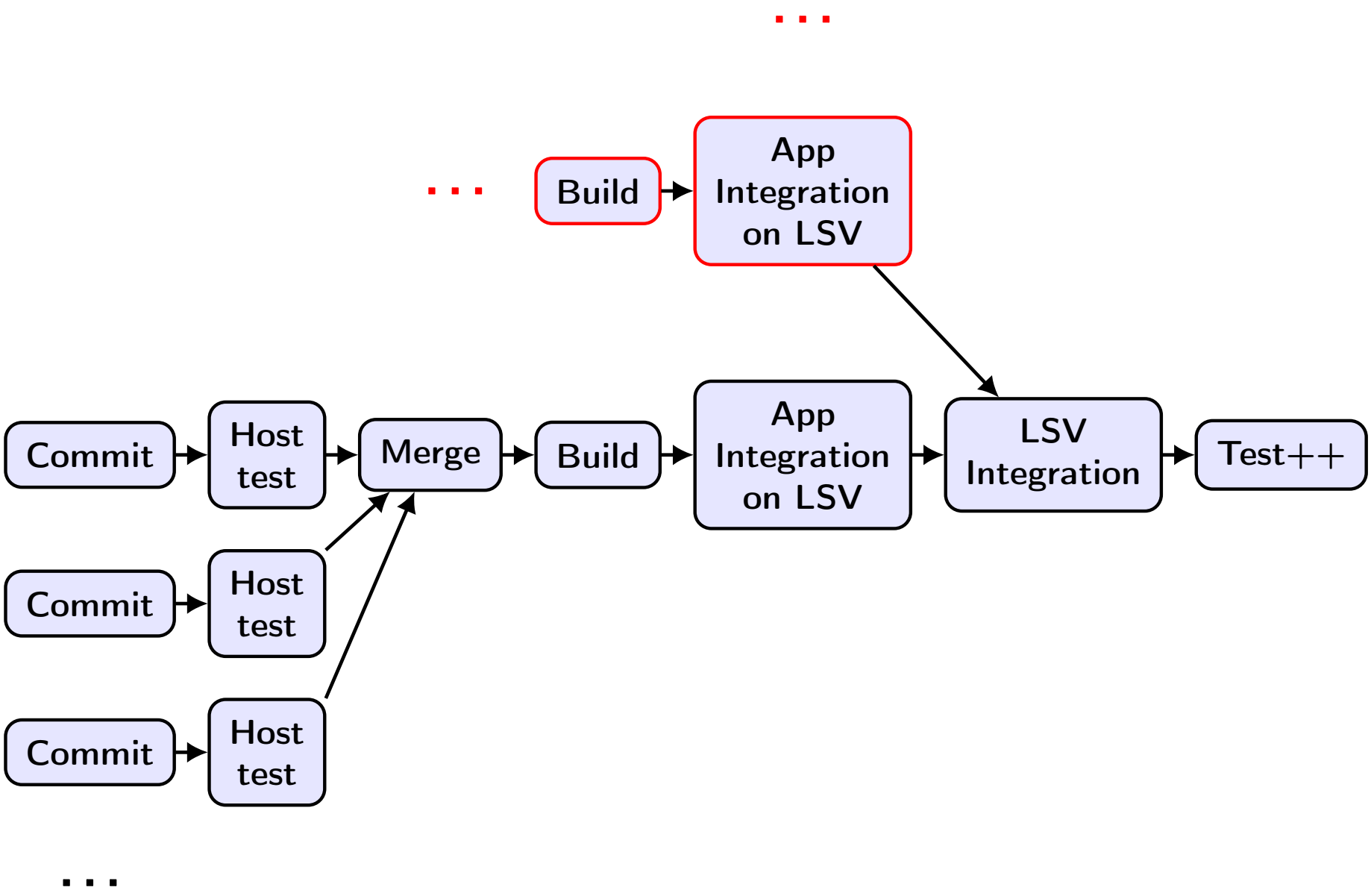
Modularization



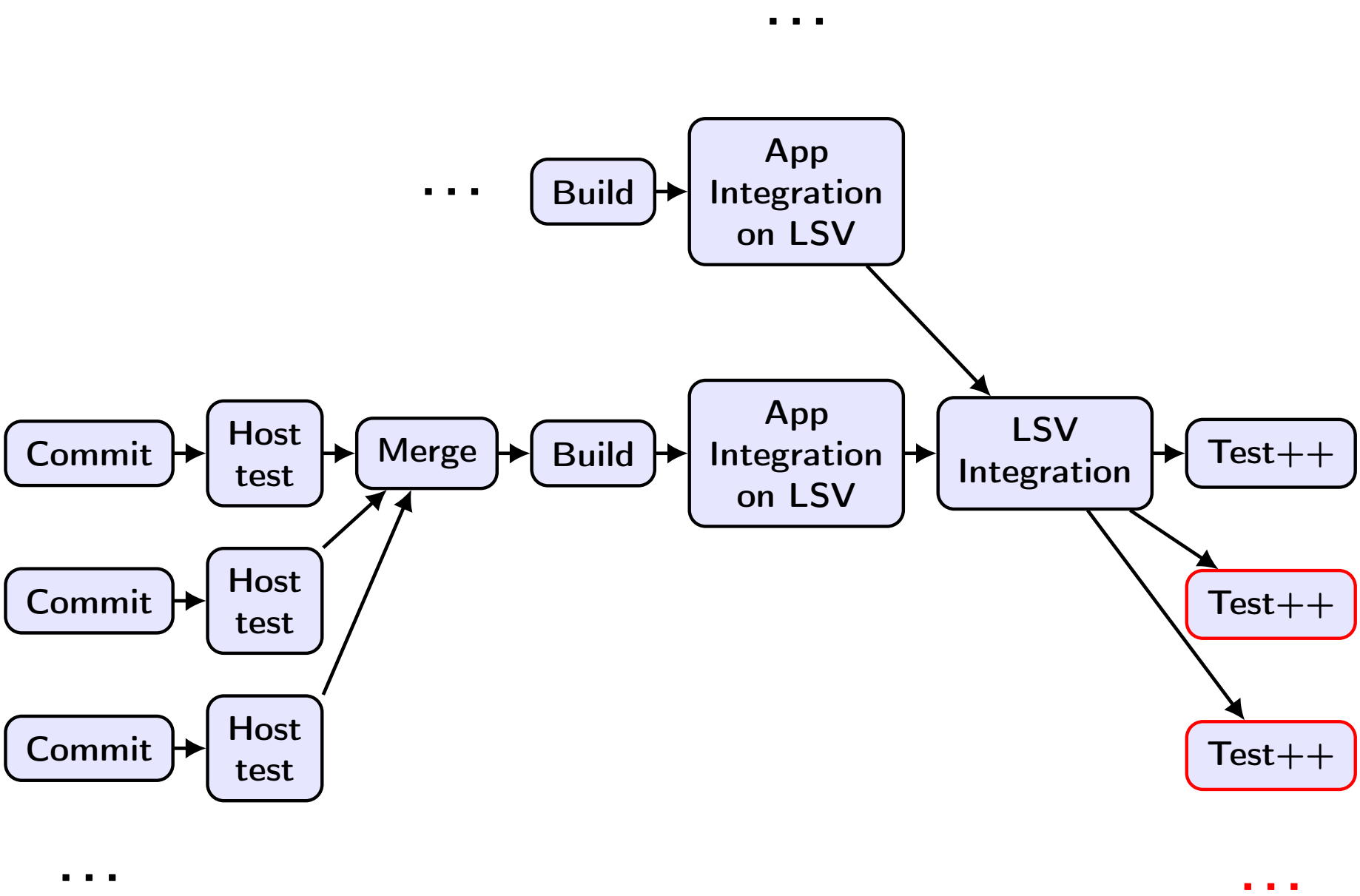
Modularization



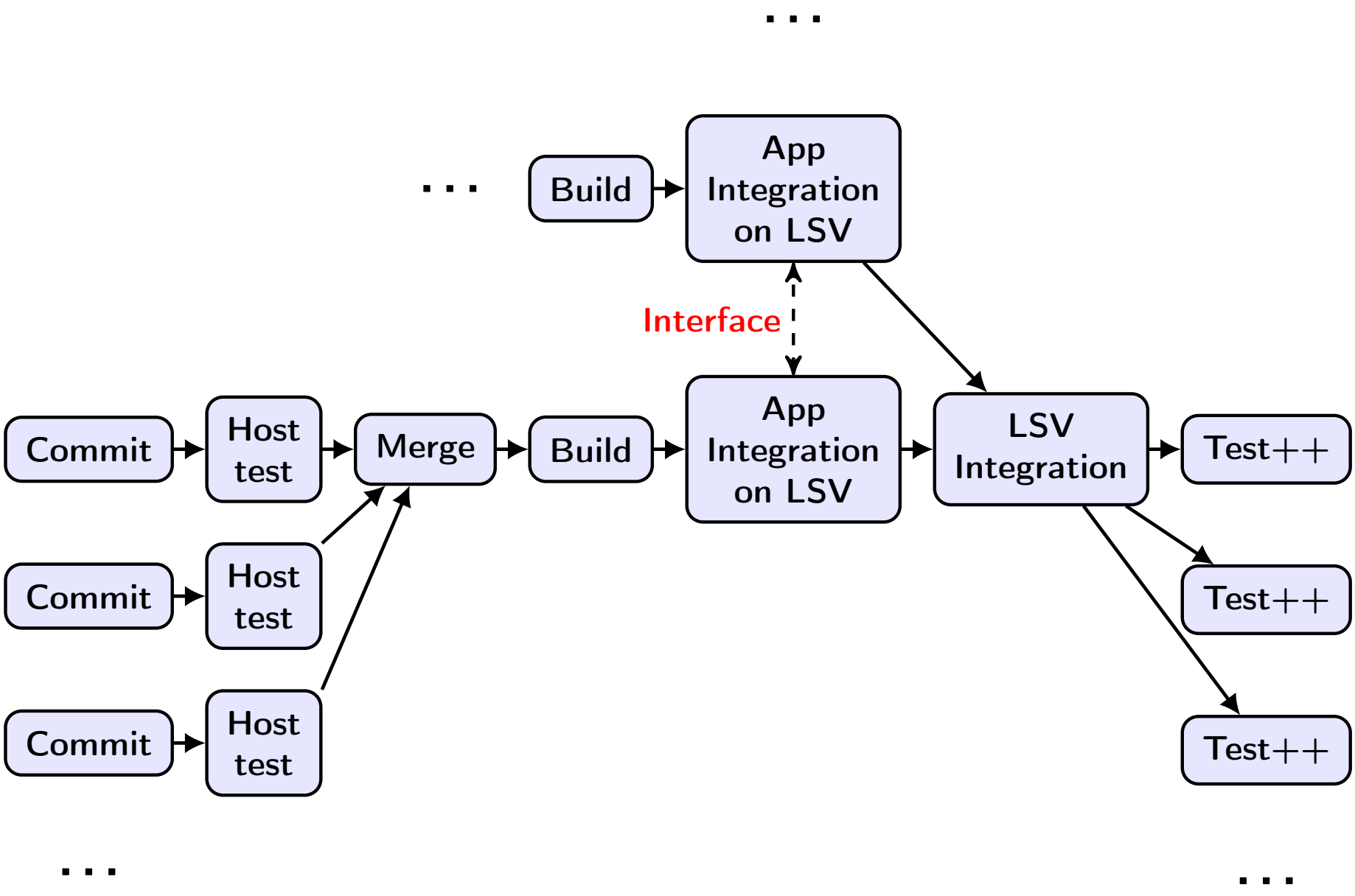
Modularization



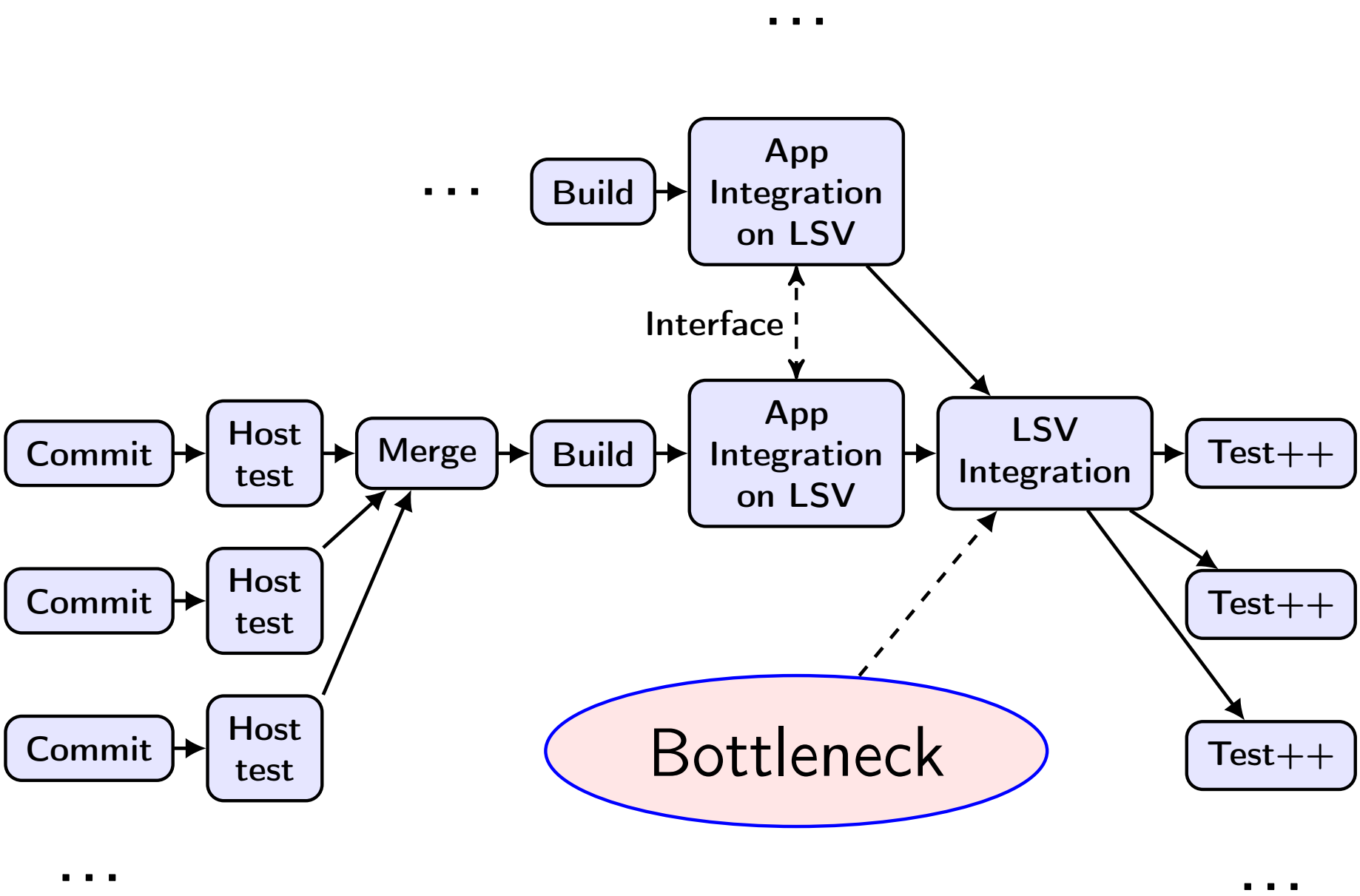
Modularization



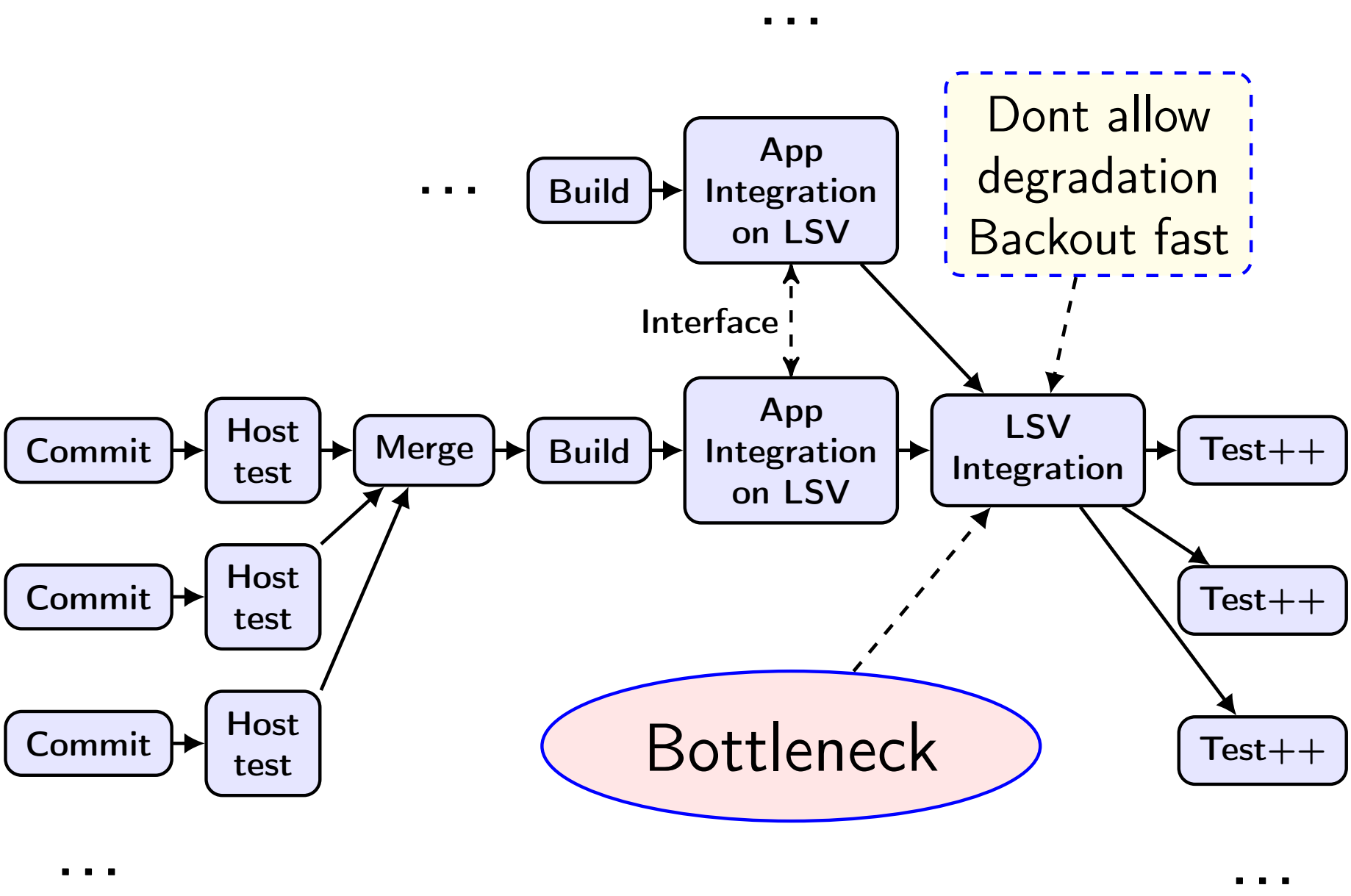
Modularization



Modularization



Modularization



Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later

- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization cont

- One developer/app should not stop flow for all
  - Bad quality → You dont get to play
  - Revert/recover first, fix later
- Needed:
  - Clean interfaces
  - Requirements
  - I.e. good architecture!
- (Enabler of Agile!)
  - More defined "sub" responsibilities, better backlogs
  - Sub-orgs solve similar problems → best solution wins!
- Shift-left principle
  - Tests involving only one application tested in app integration
  - Move tests that often fail earlier in the loops

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"

- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Modularization bad things

- More spread out
  - Harder to cooperate
  - Multiple solutions to same problem (alignment)
- Permissions
  - "*Why should you have access to my code?*"
- "Box thinking"
  - "My box is perfect" $\rightarrow$ someone elses problem
  - Remember: All working for same goal

Big CI Problems

- Test scope size
  - Can we run all tests?
  - Where should tests run?
  - Are all tests passing?
- Tracking
  - Where is my commit?
  - Is my commit ok?
- Intermittency
  - Lots of tests + intermittent tests $\equiv$ no flow
- Lead time
  - Feedback loop
- Many developers
  - $\rightarrow$ Many Bottlenecks
  - Dependencies (expected and unexpected!)

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Way towards Big CI, early actions

1. Parellelism
   - Enables running many tests
2. Build avoidance / caching
   - Don't rebuild source/objects that have not changed
   - Cache objects/build dependencies between consecutive runs
3. Smart testing
   - Many tests $\longrightarrow$ running all cripples CI
   - Only run tests that are related to change
4. Invest in Application and CI architecture
   - Design for testability
     - Divide application into sub responsibilities (modularization)
     - Communicate with backwards compatible interfaces
     - Separation of concerns!
   - Mocking!

Running all tests

```
$ cd project-x
$ . ci/setup.sh
$ time apps/app00/test/test.sh
## Running tests for /home/solarus/projects/project-x/apps/app00
# Doing complicated arithmetic (aka sleeping) for 8 seconds ...
# Done!
real  0m8.014s
```

```
$ time find -name test.sh -exec {} \;
## Running tests for /home/solarus/projects/project-x/apps/app04
# Doing complicated arithmetic (aka sleeping) for 0 seconds ...
# Done!
...
## Running tests for /home/solarus/projects/project-x/apps/app03
# Doing complicated arithmetic (aka sleeping) for 28 seconds ...
# Done!
real  11m13.586s
```

Running all tests

```
$ cd project-x
$ . ci/setup.sh
$ time apps/app00/test/test.sh
## Running tests for /home/solarus/projects/project-x/apps/app00
# Doing complicated arithmetic (aka sleeping) for 8 seconds ...
# Done!
real  0m8.014s
```

```
$ time find -name test.sh -exec {} \;
## Running tests for /home/solarus/projects/project-x/apps/app04
# Doing complicated arithmetic (aka sleeping) for 0 seconds ...
# Done!
...
## Running tests for /home/solarus/projects/project-x/apps/app03
# Doing complicated arithmetic (aka sleeping) for 28 seconds ...
# Done!
real  11m13.586s
```

Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

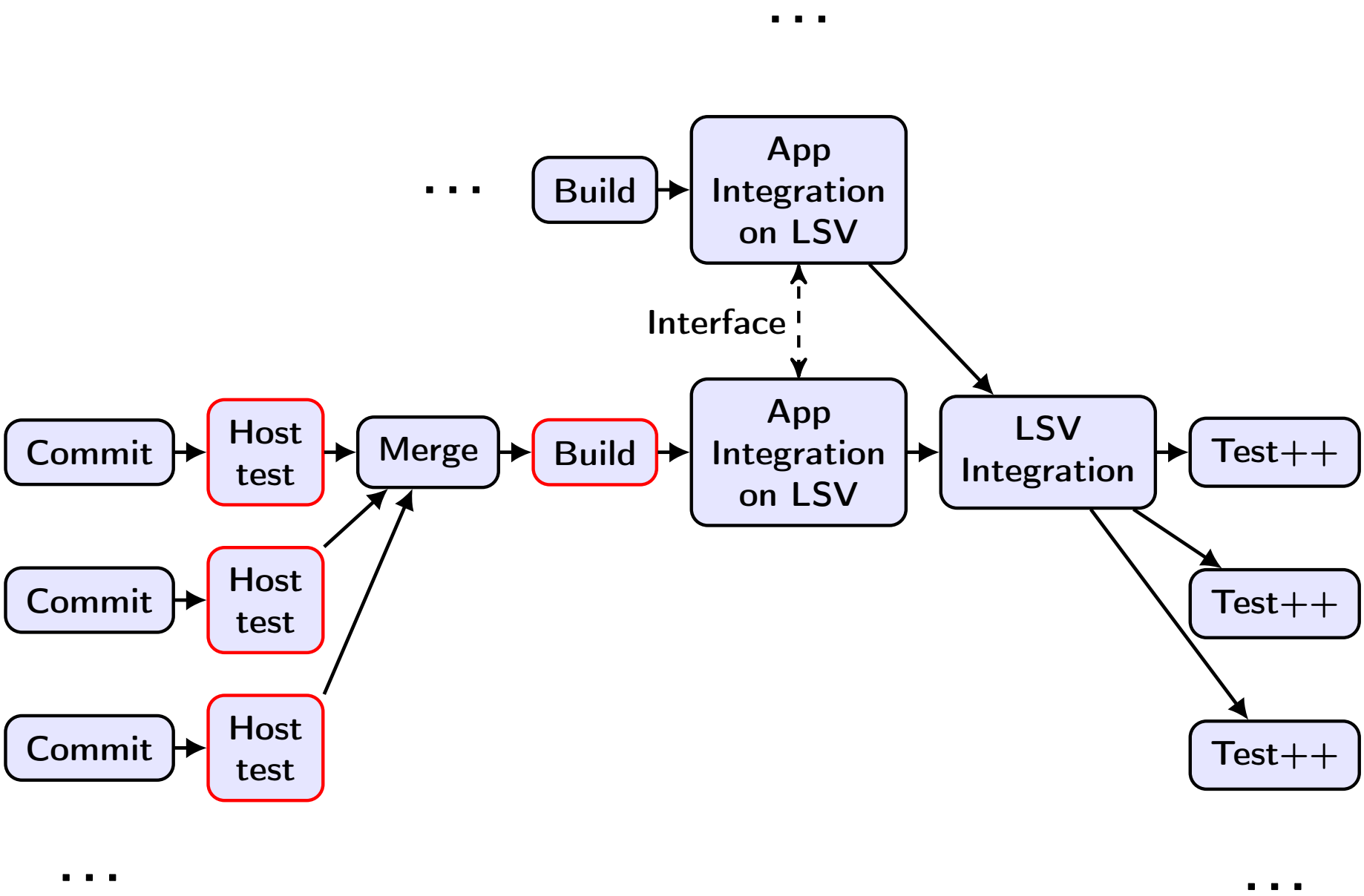
Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

Running all tests

- In this case 50 suites
  - Around 15 seconds to finish $\longrightarrow$ on average 12.5 minutes running sequentially
- Example from one repository:
  - 1 929 test suites
  - (1 035 437 lines of test code)
- Around 15 seconds to finish $\longrightarrow$ about 482 minutes of sequential run time
  - I.e. a work day...

Scalability, a practical example



Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

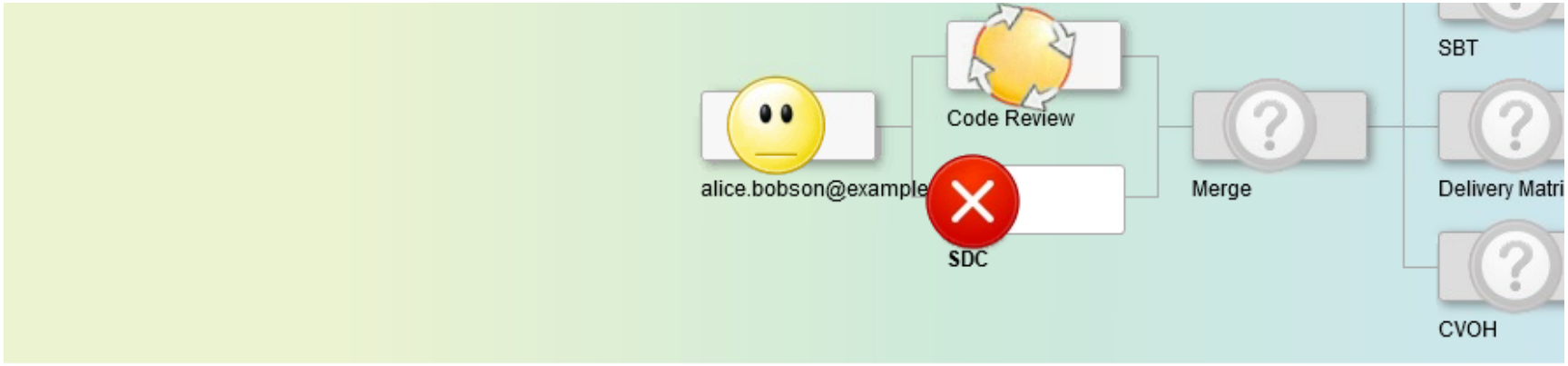
Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Big CI, a practical example

- Parallelism
  - Run as many test as possible at the same time
- Smart testing
  - Run only tests related to the application/change
- Build avoidance
  - Cache build artifacts that has not changed
  - `ccache`
- Dynamic, developer driven, test scope
  - Everyone should be able to add/remove test cases from gating guard
  - Demystifies CI, less scary
  - Developers are not relying on CI teams for guard update
- Developer feedback
  - What has gone wrong
  - How can I reproduce it?!

Example of Big CI feedback



General Information CNA_SDC (What is this?)

Jenkins job	SDC
Result	Failure
Start Time	Thursday, 29th October 2020, 12:38:14
End Time	Thursday, 29th October 2020, 12:52:04

cc.crbs.nc.ucBl.coordinatorsSwU.bblsResHandlerCsSubS

File Edit Format View Help

```
To reproduce:

git clone .... repo
cd repo
source ci/setup
cd cc/crbs/nc/ucBl/coordinatorsSwU/bbIsR
kp5 -ts coverage
```

Unix

Execute Suites information

Show all 42 suites

unit	clean	codechecker	coverage	memtest	run	staticCheck
cc/crbs/nc/ucBl/coordinatorsSwU/bblsResHandlerCsSubSwU/test/bin		✓	✗	✗		✓
cc/crbs/nc/ucBl/deviceAdaptersSwU/ricmDeviceCtrlCsSubSwU/test/bin	✓	✗	✗		✗	✓

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

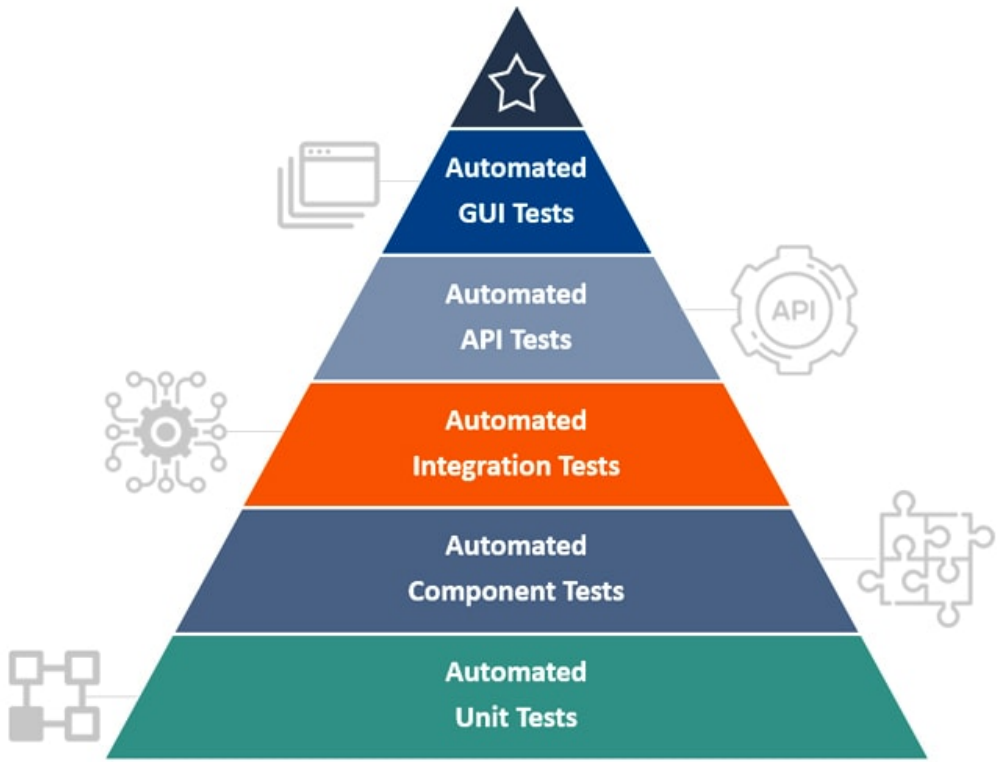
1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   -
4. Log
   -
   -
5. Sta
   -



olications

ystem

mplete CI

in

   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Way towards BIG CI, later actions

1. Modularization (architecture)
   - Integration Testing of subset of complete application
2. Layered testing
   - E.g. run long running system testing later
3. CI for CI
   - Run CI changes in same delivery/test flow as applications
4. Logging (big data)
   - Save test results / data of test case executions!
   - Impossible for humans to understand complete system
     - Understanding individual problems is easier
     - Track where each specific problem occurs in complete CI automatically
5. Stability and Recovery
   - Not running tests in later loops is expensive!
     - Long feedback from commit to test execution
     - If one test fails also following tests $\longrightarrow$ big gap in testing feedback until problem solved
   - Automatic recovery if application / tests misbehave

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test e
- Hard to guarantee stand
- Solution: Tests try to re nown working configuration be
  - Never give up!
- Without recovery, tests dback as well
  - One test failure migh part of system
  - $\longrightarrow$ Other product/t len

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability and Recovery

- Problem: Testing on real hardware $\longrightarrow$ tests or product faults might break test environment
- Hard to guarantee stand alone testing

- Solution: Tests try to recover environment to known working configuration before continuing
  - Never give up!
- Without recovery, tests after tc failure loose feedback as well
  - One test failure might lower confidence in big part of system
  - $\longrightarrow$ Other product/test issues potentially hidden

Stability: No recovery

```
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from ComplicatedApplication
[ RUN      ] ComplicatedApplication.Test1
[       OK ] ComplicatedApplication.Test1 (15 min 32s)
[ RUN      ] ComplicatedApplication.Test2
[  FAILED  ] ComplicatedApplication.Test2 (10 min 12s)
[ RUN      ] ComplicatedApplication.Test3
[  FAILED  ] ComplicatedApplication.Test3 (3 min 5s)
[ RUN      ] ComplicatedApplication.Test4
[  FAILED  ] ComplicatedApplication.Test4 (3 min 10s)
[ RUN      ] ComplicatedApplication.Test5
[  FAILED  ] ComplicatedApplication.Test5 (3 min 30s)
[----------] 5 tests from ComplicatedApplication (0 ms total)
[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (0 ms total)
```

Stability: With recovery

```
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from ComplicatedApplication
[ RUN      ] ComplicatedApplication.Test1
[       OK ] ComplicatedApplication.Test1 (15 min 12s)
[ RUN      ] ComplicatedApplication.Test2
[  FAILED  ] ComplicatedApplication.Test2 (10 min 30s)
[ RECOVER  ] ComplicatedApplication.Test2 (2 min 3s)
[ RUN      ] ComplicatedApplication.Test3
[       OK ] ComplicatedApplication.Test3 (10 min 15s)
[ RUN      ] ComplicatedApplication.Test4
[       OK ] ComplicatedApplication.Test4 (5 min 48s)
[ RUN      ] ComplicatedApplication.Test5
[       OK ] ComplicatedApplication.Test5 (28 min 13s)
[----------] 5 tests from ComplicatedApplication (0 ms total)
[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (0 ms total)
```

Test failure

- Test case fail $\rightarrow$ What do you do?

Test failure

- Test case fail $\rightarrow$ What do you do?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...

- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging

- Remember Big CI Problems:
  - Many tests+developers+apps/Tracking/Intermittency...
- Test failed in App Integration $\rightarrow$
  - Test failed before? (same way!)
    - In same App/other apps?
    - On certain configurations?
  - Intermittent?
    - More intermittent today than last week?

Logging cont

- Without data, we are blind to degradations

- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Logging cont

- Without data, we are blind to degradations
- Solution: automatic result tracking!
  - Test failure messages, configurations, target log analysis
  - *I.e. store test results and test meta data*
- Tool to automatically tag knows faults in stored results
  - Tagged faults can be visualized separately (ticket)
  - $\longrightarrow$ easier to understand
  - Know if fix helped without reading single test log!

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Summary

- Parallelism, smart testing, caching crucial
- Developer driven test scope (dynamic scope), reproducibility
- Modularization
  - Test scope per sub-application
  - Avoids bottlenecks
  - Good architecture (CI + application)
- Logging (big data)
  - Save results for:
    - Fault tracking
    - Long term performance tracking
    - Troubleshooting

Labs

- Lab 3: Add smart testing and dynamic test scope for Project-X
- Lab 4: Set up parellelism for Project-X
- https://github.com/dev4242/project-x/tree/main/documents/labs
  - Labs, cheat sheet!

Labs

- Lab 3: Add smart testing and dynamic test scope for Project-X
- Lab 4: Set up parellelism for Project-X
- https://github.com/dev4242/project-x/tree/main/documents/labs
  - Labs, cheat sheet!

Labs

- Lab 3: Add smart testing and dynamic test scope for Project-X
- Lab 4: Set up parellelism for Project-X
- https://github.com/dev4242/project-x/tree/main/documents/labs
  - Labs, cheat sheet!

Labs

- Lab 3: Add smart testing and dynamic test scope for Project-X
- Lab 4: Set up parellelism for Project-X
- https://github.com/dev4242/project-x/tree/main/documents/labs
  - Labs, cheat sheet!

Questions?