

# Offline / Online Connection

Steffen Luitz, SuperB Workshop, SLAC, October  
2009

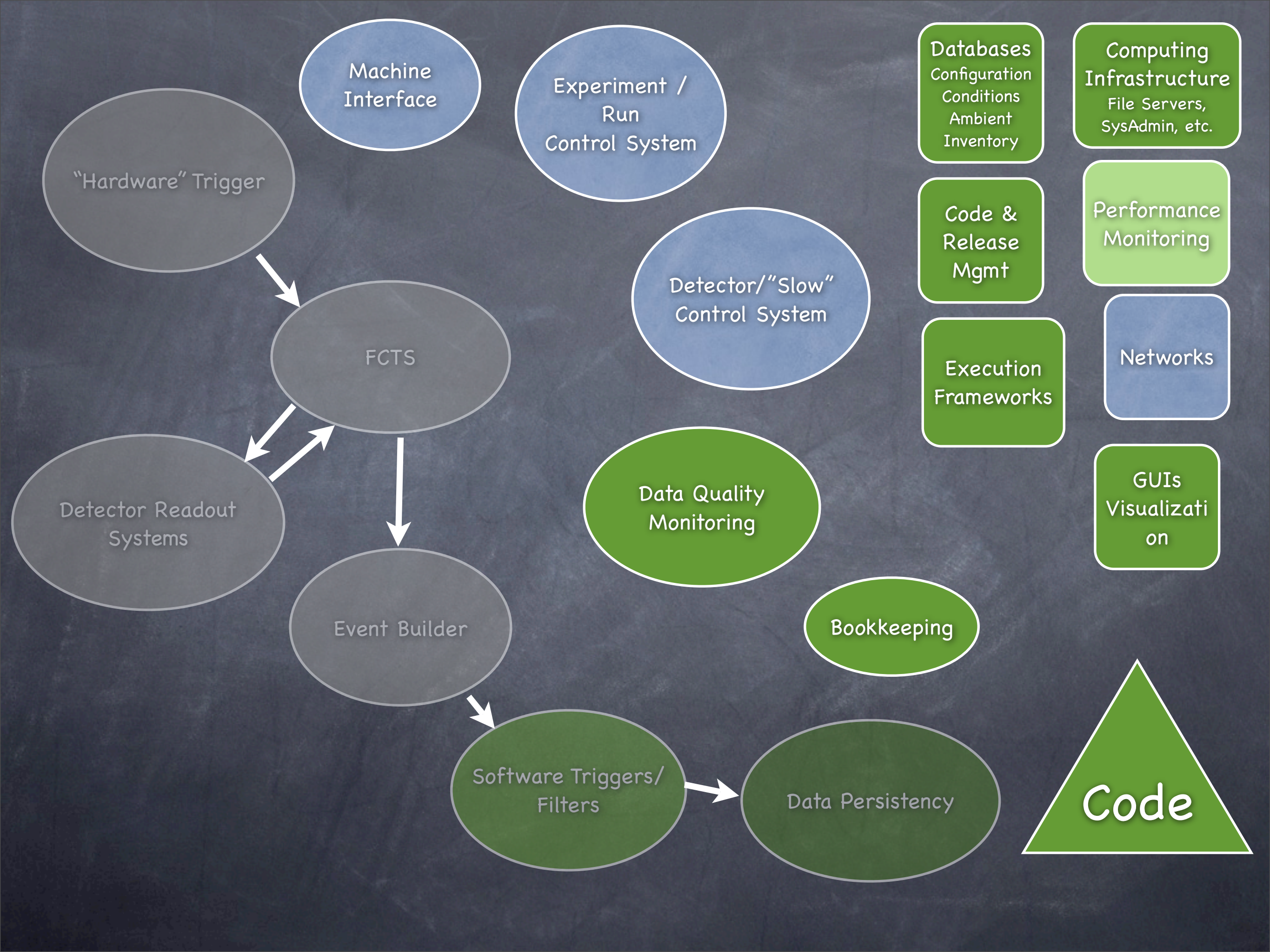


# From BaBar Experience

## Some Talking Points

- Components in Online
- Code Management, Release and Build System
- Data Format & Dependency Management
- Online vs. Offline Paradigms
- Shared Code
- Performance / Quality Control
- Bookkeeping







# Code Management, Release+Build System (1)

- BaBar

- "2 1/2" different release + build systems

- Dataflow build system (embedded + regular) - for a reason
    - "Online Releases" - using SRT but special
    - Base SRT releases

- Some of the Issues

- Impractical code dependencies (Online on top of SRT Base) and complex dependency management (what goes where, etc.). Online, where you want most flexibility & agility depended on this huge blob of SRT base
    - Search path overlays can be dangerous
    - Several attempts to improve - never high priority

- Opportunity for R&D in code organization

- Code organization (peer modules) as well as runtime (dynamic loading)
    - Some ideas are already there



# Code Management Release+Build System (2)

- Other areas of interest
  - Configurable firmware in more and more places (embedded processors, FPGA configurations, etc.)
    - Need to manage & track - What should be integrated with release mgmt?
  - Scripts, glue code, computer configurations
    - ditto
  - Especially Online needs to be able to "test" deploy and back out quickly
    - More automation than BaBar (path / symlink based scheme) seems desirable
- Design mechanisms to ensure consistency of what code is running
  - large farms - make sure every machine has the right executables and configuration



# Data Format + Dependency Mgmt

- Data format (or semantics) or database schema changes are always difficult
  - Fundamental problem: Have to deploy code that understands the new format / schema before deploying code that produces data in that format
- Unfortunate effect in BaBar: Interesting improvements in Online got delayed for months because downstream release building and deployment was difficult
- Attempt to design a system that provides the necessary agility and flexibility for Online
  - e.g. Simplify downstream release builds / deployment
  - e.g. look into how forward&backward compatibility could be achieved



# Online / Offline Paradigms

- Boundaries between "Online" & "Offline" are becoming increasingly blurry.
  - Computers are getting much faster, so you can do much more "online"
  - Code sharing is very attractive
- Fundamental difference:

Online is where you (often) can't redo things if something goes wrong!



# Shared Code

- Lots of benefits from Online/Offline code sharing
- Online paradigms impact how code used by Online must behave – especially in areas of input validation and error handling:
  - Code used in Online often needs to be very robust against malformed input data. Alert and skip, not segfault & core dump. Impacts all code used in Online.
  - In Online systems assert() is not always your friend – code needs ability choose what to do in the face of errors. Exceptions (or similar mechanisms)
  - Need to propagate errors to higher levels where meaningful decisions can be made
  - Restarting from scratch “to reset” is not always acceptable, especially if startup times are long
- True code sharing requires some thought and reliability engineering



# Performance & Quality Control

- When running in an Online context, code may run in or close to the “dead time path”
  - Careful engineering & testing
  - Take into account average & worst-case performance
  - Engineer frameworks that allow to deal with worst-case
    - e.g. “non-blocking” behavior where appropriate
- Overall performance
  - Ability to utilize underlying platform
    - Multithreading. GPU, other non-uniform architectures, etc.
- Quality control
  - verify code to standards outlined on previous slide



# Bookkeeping & Storage

- Just some thoughts
  - Over the SuperB lifetime, storage media may significantly change in characteristics (e.g. Tape, Disk, SSD, ...)
  - Decouple file / container sizes from data grouping concepts such as "Runs"
  - Allow for optimization of file sizes - splitting, merging
  - Do not introduce artificial choke points in the system
    - such as e.g. merging run parts into complete run files in BaBar
  - Is a versioning capability for raw data (e.g. in case of "manual" repairs or removal of "bad" parts) needed
    - I'm tempted to say no - forego the complexity and write-off the data
- Bookkeeping system - shared between Online & Offline must hide the complexities from users (and apps)