Job Monitoring at STAR

This presentation is the personal view of the author. It was not approved (yet) by STAR

STAR Collaboration: production since 1999

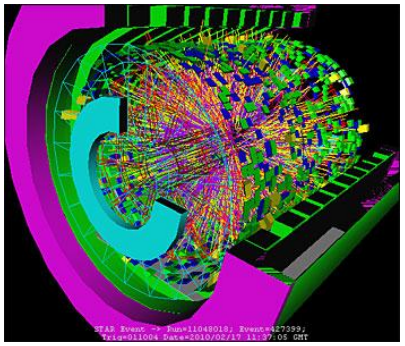
- Man power - 53 institutions from 12 countries, with a total of 524 collaborators (http://www.star.bnl.gov/central/collaboration/)
- Data volume – Run 2010 – 1 Pb DAQ data / 6 months (2010 = 10 x 2009)
- Storage – 2 HPSS equipped centers:
  - BNL - https://web.racf.bnl.gov/Facility/GCEGangliaRHIC/
  - PDSF- http://www.nersc.gov/nusers/resources/HPSS
- Computer: 2 large farms:
  - BNL/RACF -https://web.racf.bnl.gov/Facility/GCEGangliaRHIC/ https://web.racf.bnl.gov/Facility/LFGangliaRHIC/
    - CAS - CPUs ~ 3000/ 4 TB RAM/ 1 Pb disks
    - CRS - CPU ~ 2000/4 TB RAM / 1 PB disks
  - PDSF - http://www.nersc.gov/nusers/resources/PDSF
    - CPU ~ 1000/ 2 TB RAM/ 0.4 Pb disks
- Bunch of the regional small centers to serve their local researchers
- “Cloud” computing.

The Types of the Software Jobs / Users

1. Calibration (STAR wide)
2. Simulation (STAR wide)
3. Reconstruction (STAR wide)
4. Analysis (PWG wide)
5. Interactive analysis (PWG)
6. Regression tests (aka nightly)
7. Custom reconstructions (researchers and institutions)
8. Custom analysis (everybody can do that)

Data flow

RHIC/STAR



Jobs / Users are split by two major centers:

BNL

BNL/HPSS

PDSF

PDSF/HPSS

and dozen of the smaller ones:

. . .

(Decentralized)Monitoring
The monitoring of the users' jobs is the job of the monitoring users

Historically all 8 types of users were to build their own monitoring systems if any.

That leaded to the waste of workforce due duplications.

Lesson: The ordinary user can not monitor his/her job alone , the framework support is needed.

The common main tools to facilitate the job monitoring were built over time (As the demand had being growing up):

1. STAR File Catalog (perl scripts)
2. SUMS "STAR Unified Job Submission Utility – http://www.star.bnl.gov/public/comp/Grid/scheduler - Java applications
3. UCM – C++ library to back C++/Fortran/Java applications
4. Web interface / Web service – to access the Accumulated information

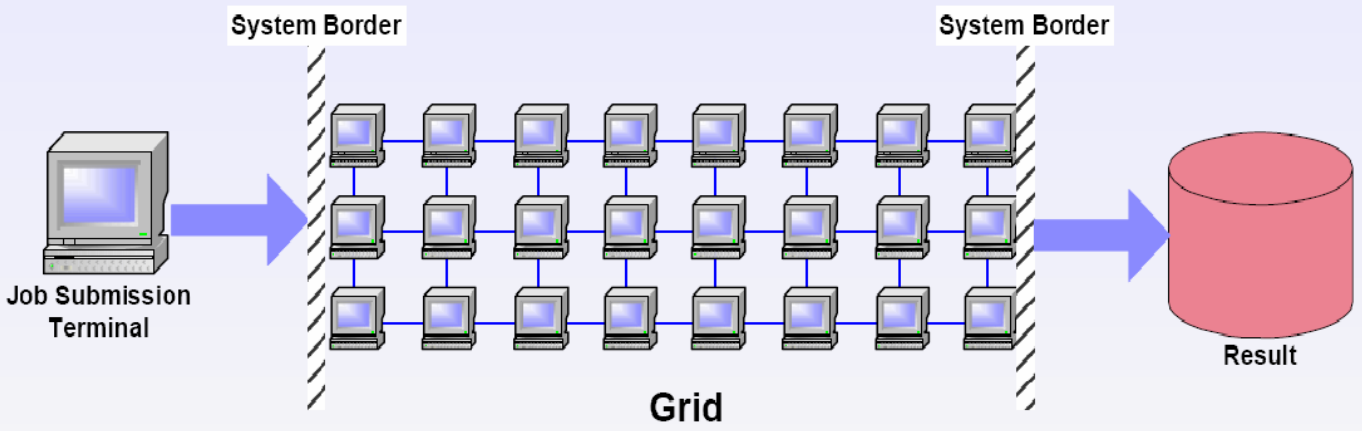
STAR Monitoring Tools:

- The tools provided by RACF/PDSF centers.
- Set of submissions SHELL scripts "grep"ing the output logs of the jobs to populate the production data based (use "File Catalog") – Production, regression nightly tests.
- File Catalog + SUMS – submit the jobs to the nodes were the data files are local. That requires the data move monitoring rather the monitoring of the jobs.
- UCM – real "job monitoring" (SBIR (DOE) funded project made by Tech-X and BNL. Tech-X contact and PI D.Alexander, BNL contact and PI J.Lauret).

“Black” hole

Why User Centric Monitoring

Grid Systems



Characteristics of Grid Systems

- No user intervention
- Blackbox for users in the job running

STAR Users on 24/7 Job monitoring shift

- Chronic lack of resource to process data encourages the users to watch the process to make sure the wise usage of the existing facilities. There is no need for 24/7 shifts.
- STAR – the users watch his/her jobs voluntary and report any inconsistency immediately .
- However, the ordinary user can not monitor his/her job alone , the framework support is needed.
- **Lesson:** Users reports the "observations". To find and fix the cause of the complain the time and forensic skill are needed or "dedicated monitoring tools and facility" to address and prevent the issues quickly

UCM - User-Centric Monitoring

User-Centric Monitoring concentrates on Job, Task & Application monitoring for a given user.

- Includes “messages” from applications unlike many other systems.
- Aggregates information for various parts of the workflow (job definition, submission, scheduling, application contexts)

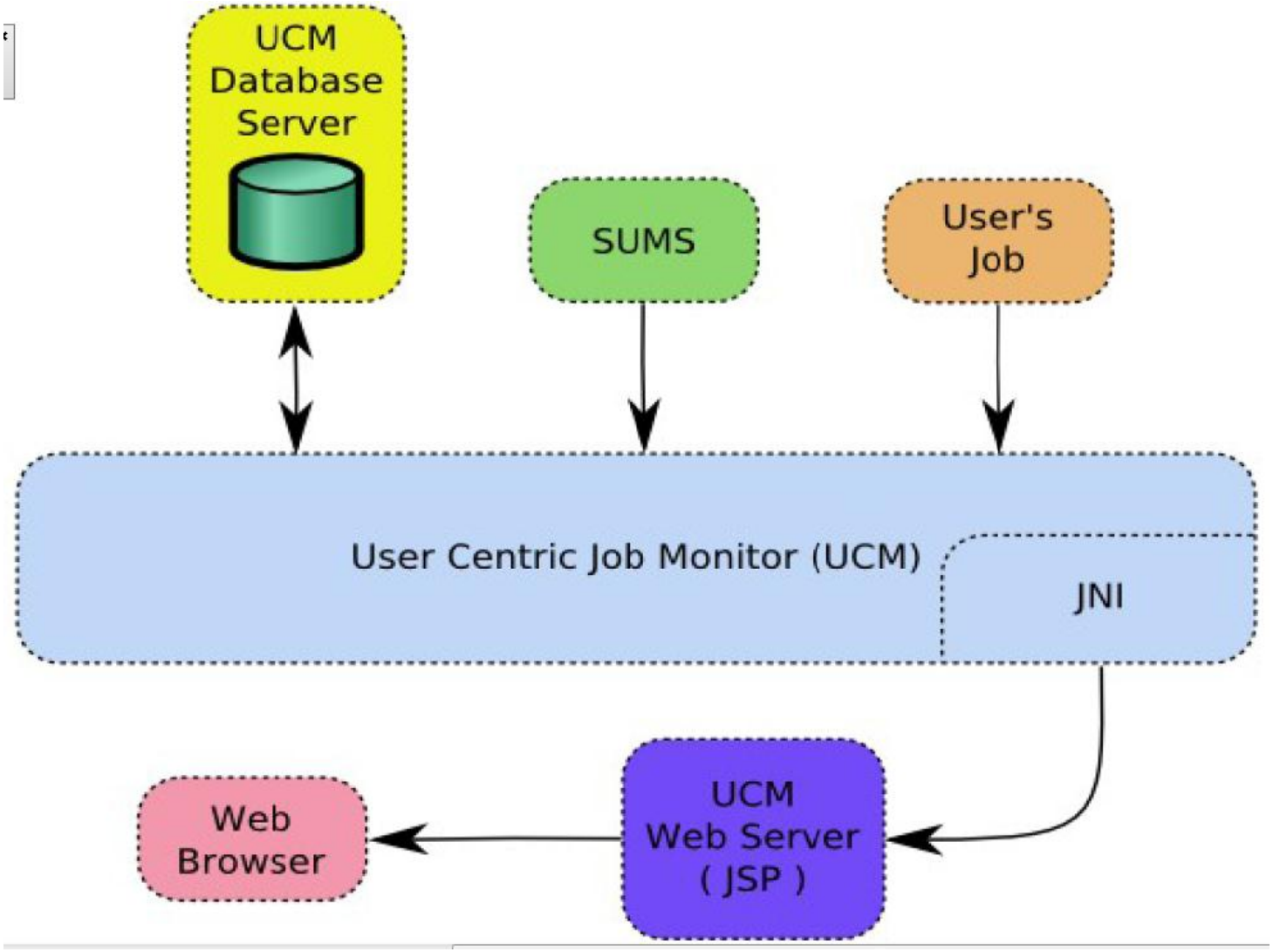
Design Requirements (aka “Constrains”)

The UCM infrastructure is built around of STAR MySQL server / Tomcat Web server and the abstract C++ interfaces.

1. Db design
2. Software design

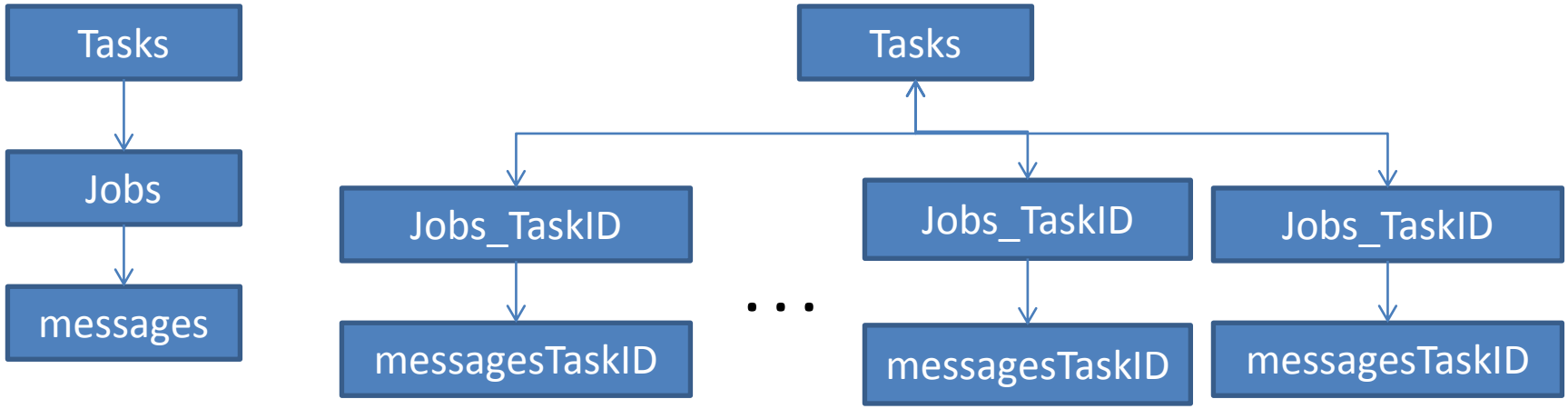
- **No Monitoring Db problem** should affect the job performance
- **No interruption of the job** <–> Db communication problems should not affect the Db integrity. – The lost of messages is the anticipated cost of doing business.
- **No “UCM” evolving”** should affect the users’ job flow (i.e. the “user” should not be asked to change his/her code because of any change of the monitoring facility
- **The layer should be useful for the end–user too**. I.e. any portion of the user job (either the “Shell script” or “Fortran written simulation” should be able to emit the “monitoring” messages

Solution:


UCM
Database
Server
SUMS
User's
Job
User Centric Job Monitor (UCM)
JNI
Web
Browser
UCM
Web Server
(JSP)

Hierarchical Db Design

- We ended up with the simple scalable Db design:
  (STAR “task” == ATLAS “jobset”)



- One may realize that the schema above is quite generic.
- As soon as one introduces the special “system” tasks/jobs it becomes capable to cover the wide range of the “system” needs (like “harvesting” the statistic across jobs and tasks)

Db “Message” table

- **`Timestamp`**
- **`JobID`** (index from the JOB table)
- **`Message type`** (enumerate)
- **`Message key`** (arbitrary string)
- **`Message value`** (arbitrary string)

Examples:

Key=“**`CPUTime`**” ->Value=“**`201.3`**” (secs)

Key =“**`RAM`**” -> Value=“**`234.5`**” (Mgb)

and

Key=“**`Average CPUTime`**” ->Value=“**`204`**” (secs)

Key =“**`Average RAM`**” -> Value=“**`240`**” (Mgb)

fits the schema too.

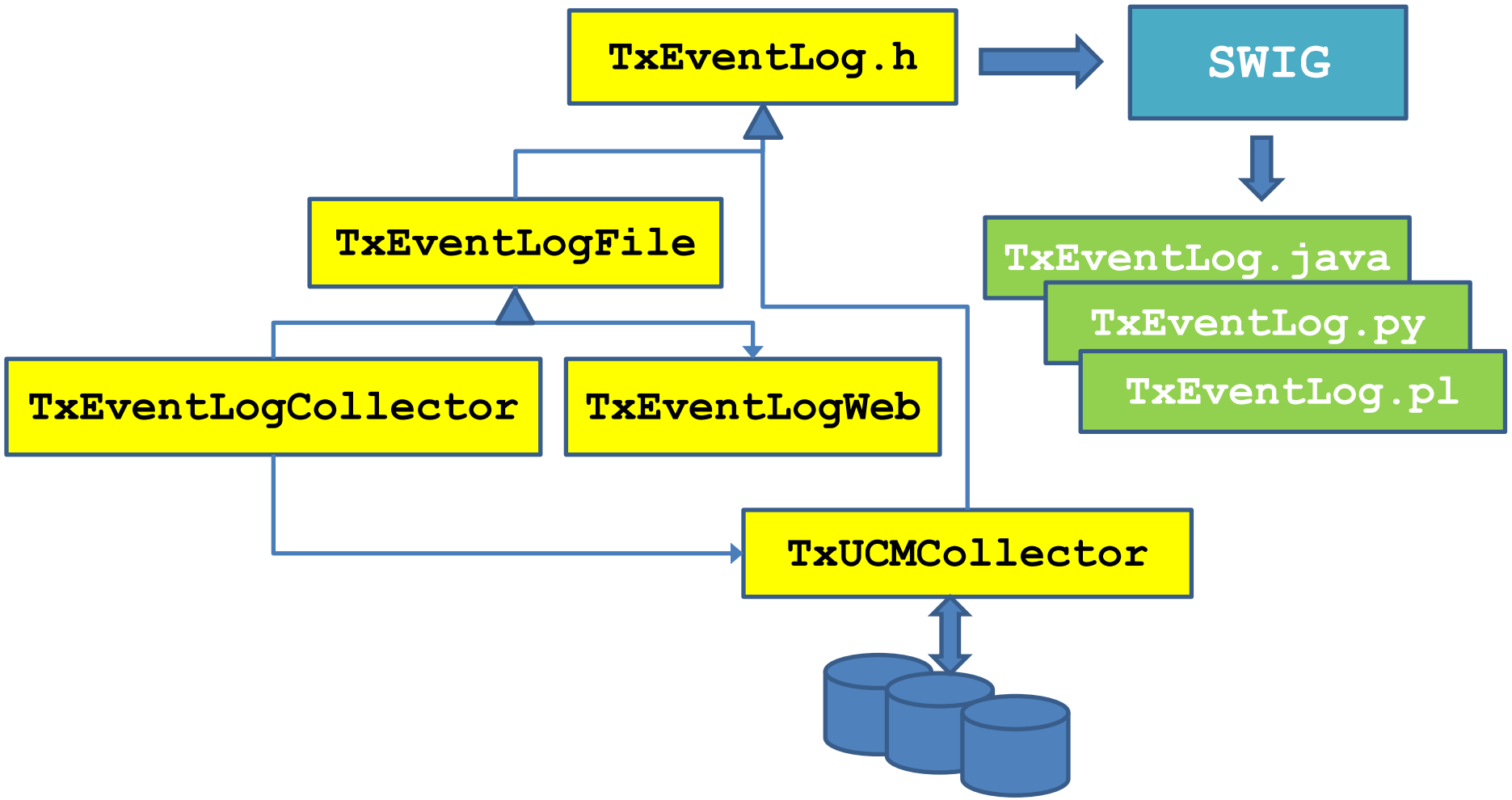
UCM – Software design

There are 3 clients:

- STAR Offline applications
- STAR Scheduler (aka SUMS)
- STAR Tomcat Web server (fc3.star.bnl.gov)

All clients communicate the Db via the abstract interface and its implementations tailored to satisfy the concrete clients

C++ → Java / Python / Perl



Example: C++ vs. Java nterfaces

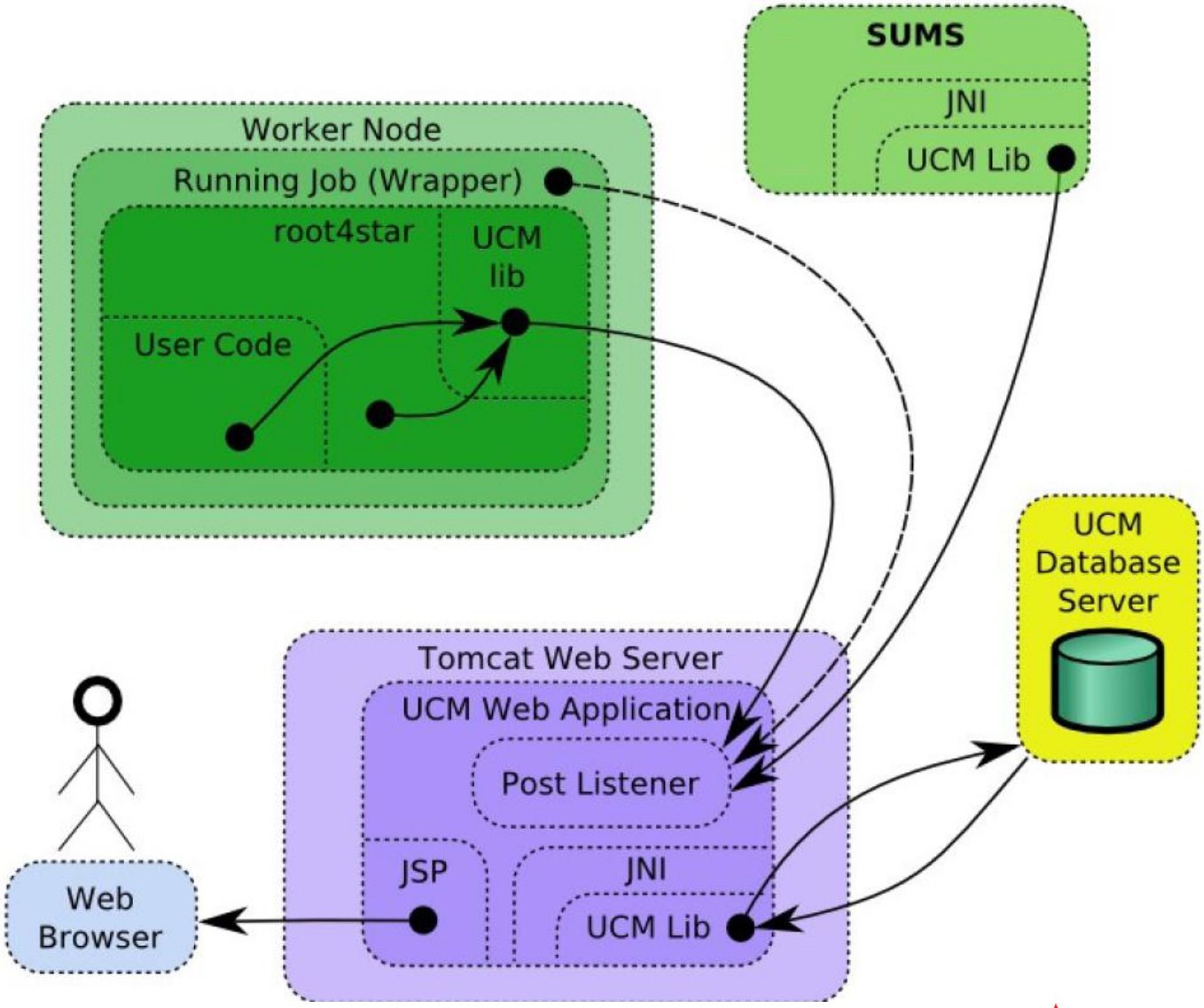
C++

```
int main(int argc , char *argv[]){
   TxEventLog &ucm =
              *TxEventLogFactory::create("ucm");
   const char *username = "starreco";
   ucm.setRequesterName(username);
   StUcmTasks &tasks =  *ucm.getTaskList(10);
   Iterator task = tasks.taskIterator();
   StRecord *r = 0;
   if (task.hasNext()) {
     r = task.next();
     StUcmJobs &jobs = *ucm.getJobList(r,20);
     Iterator  job = jobs.jobIterator();
     while (job.hasNext() ) {
       r = job.next();
       r->printHeader();
       r->print();
       StUcmEvents &events=
            *ucm.getEventList(r,20);
       Iterator event =
             events.eventIterator();
       while(event.hasNext() ){
         r = event.next();
         r->print();
       }
     }
. . .
```

Java

```
class testUcm {
  public static void main(String [] argv) {
    TxEventLog ucm =
           TxEventLogFactory.create("ucm");
    String username = "starreco";
    ucm.setRequesterName(username);
    StUcmTasks tasks =  ucm.getTaskList(10);
    Iterator task = tasks.taskIterator();
    StRecord r=null;
    if (task.hasNext()) {
      r = task.next();
      StUcmJobs jobs = ucm.getJobList(r,20);
      Iterator  job = jobs.jobIterator();
      while (job.hasNext() ) {
        r = job.next();
        r.printHeader();
        r.print();
        StUcmEvents events=
             ucm.getEventList(r,20);
        Iterator event =
             events.eventIterator();
         while(event.hasNext() ){
           r = event.next();
           r.print();
         }
    }
. . .
```

UCM UI via Web Frontend



UCM – scalable (?) solution

- **"Functional scalability"**
  - any new type of information can be added without breaking the developers' legs.
  - The real functionality is built with C++ code leveraging the C++ robustness, performance, code re-use. C++ is the only interface @ STAR that the end-user C++ applications can use. Therefor STAR had no other choice.
  - Java/Python/Perl interfaces are built automatically leveraging the script languages flexibility and quick customization.
- **"Performance scalability"**
  - adding some extra information to the customer demand does not affect the other jobs / customers.
  - All Db tables are relatively small hence the retrieving information for any particularly job ought to be fast. Easy to introduce the distributed "solution" - each job/message table for the concrete task can be allocated the separate Db / node / machine / location if needed.

Conclusion (aka “Lessons”;-)

- The framework support is essential.
- Stable simple abstract (C++) API helps significantly . It features no distraction for the “client code” and “freedom” for the developers to polish things and meet the new challenges.
- Automatically conversion to the other languages interfaces helps to minimize the workforce to maintain the monitoring framework, reducing the costly future “surprises”.
- “Real time” job monitoring is required.
- Hierarchical Db table organization to provide the fast access.
- Stable abstract Db schema for the low level layers to allow flexibility.
- Standard Web server (TomCat) to satisfy the “future” requirements

Acknowledgement

Special thanks:

Jerome Lauret -- STAR Computing leader,

Lidia Didenko – STAR Production Manager

Questions ?