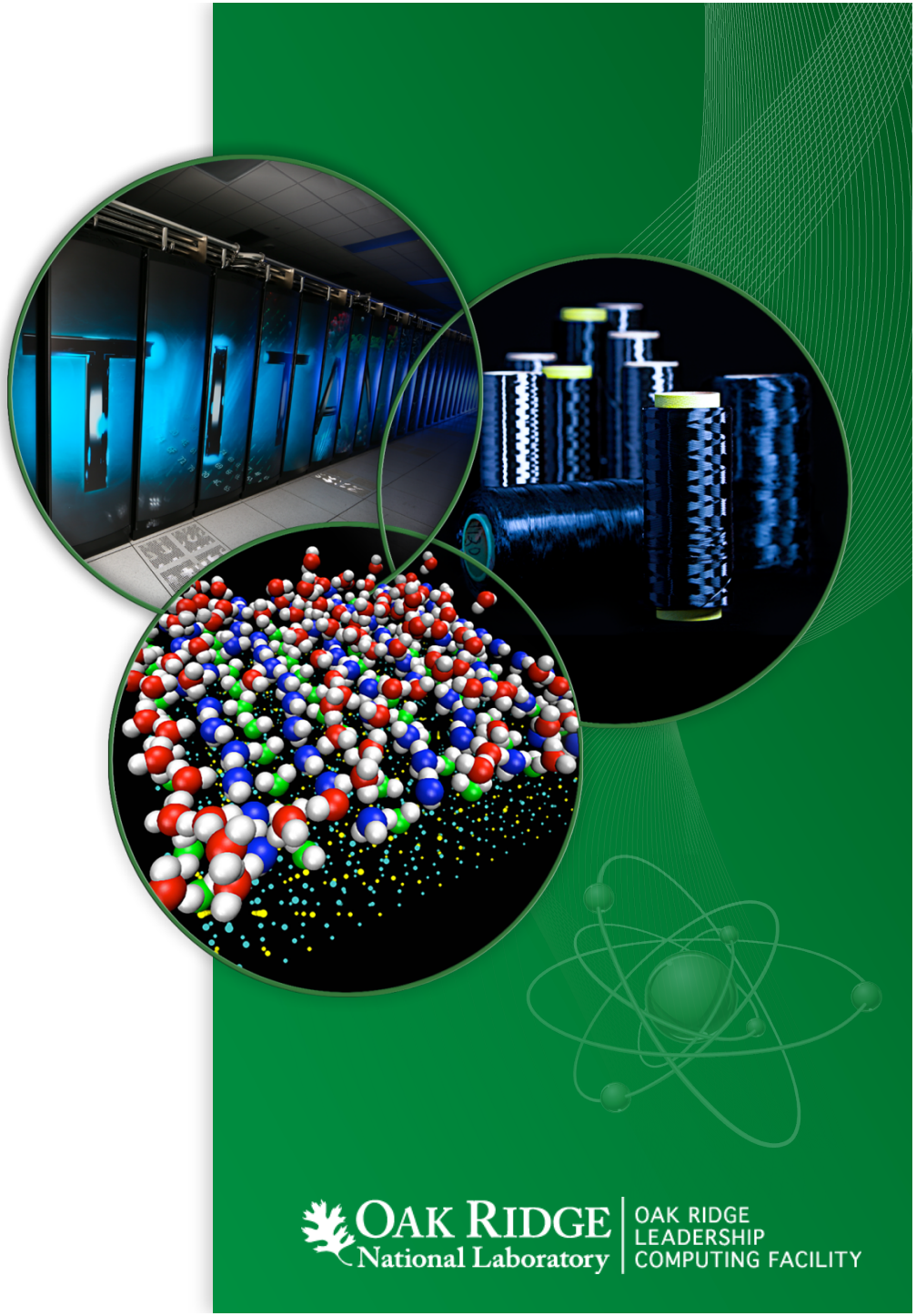
Designing Service-Oriented Tools

for HPC Account Management and Reporting

Adam G. Carlyle
Robert D. French
William A. Renaud



ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Projects: OLCF and NCRC

Outline of Topics

- Tenets of service-orientation
- Case: non-service-oriented application
- Shortcomings of non-service-orientation
- Case: redesign using service-oriented principles
- Benefits of service-orientation
- Service plugins
- Reporting as a service

Tenets of Service-orientation

(4) Tenets of Service-Orientation

- "Service" = modular, reusable software component to carry out a business function of an organization
- No industry standards that exactly define service-orientation
- Microsoft's Don Box - Simple Object Access Protocol (SOAP) and SOA:
  1. Service boundaries are explicit
  2. Services are autonomous
  3. Services share schema and contract, not class
  4. Service compatibility is determined based on policy

(4) Tenets of Service-Orientation

1. Service boundaries are explicit
   - Careful control over communication channels exposed
2. Services are autonomous
   - Individual data stores and support infrastructure
   - Assume all incoming message data could be malformed and/or malicious
3. Services share schema and contract, not class
   - Consistent APIs, not just common data structures
4. Service compatibility is determined based on policy
   - machine-readable policy statement lists capabilities

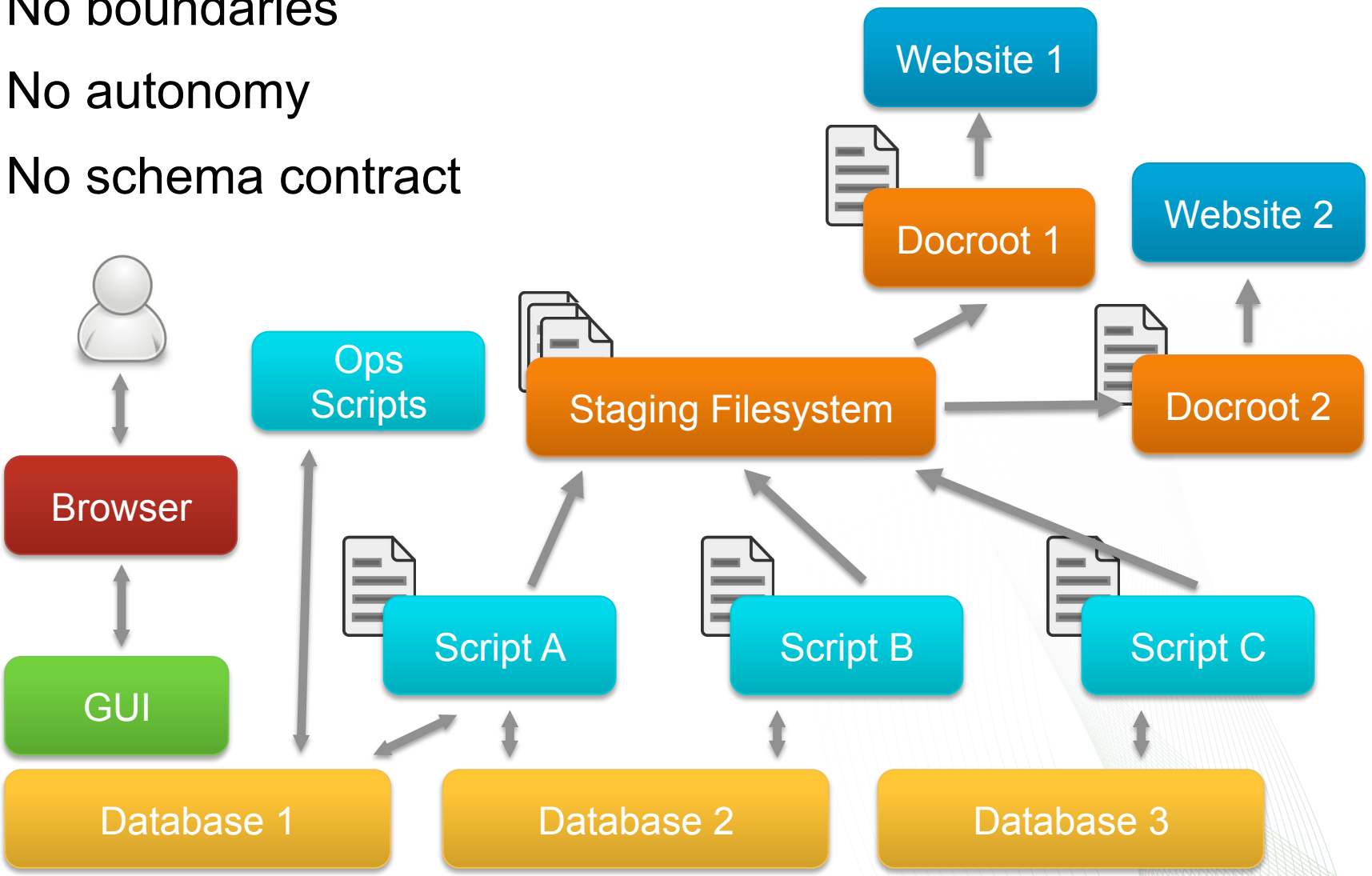
Case: A Non-service-oriented Application

Downtime Tracking System

- Currently provide slick auto-downtime notifications
- Want to provide more comprehensive downtime info
- Want bookmark-able center status page on web
- But don’t want to store info in web CMS

Downtime Tracking System

- No boundaries
- No autonomy
- No schema contract



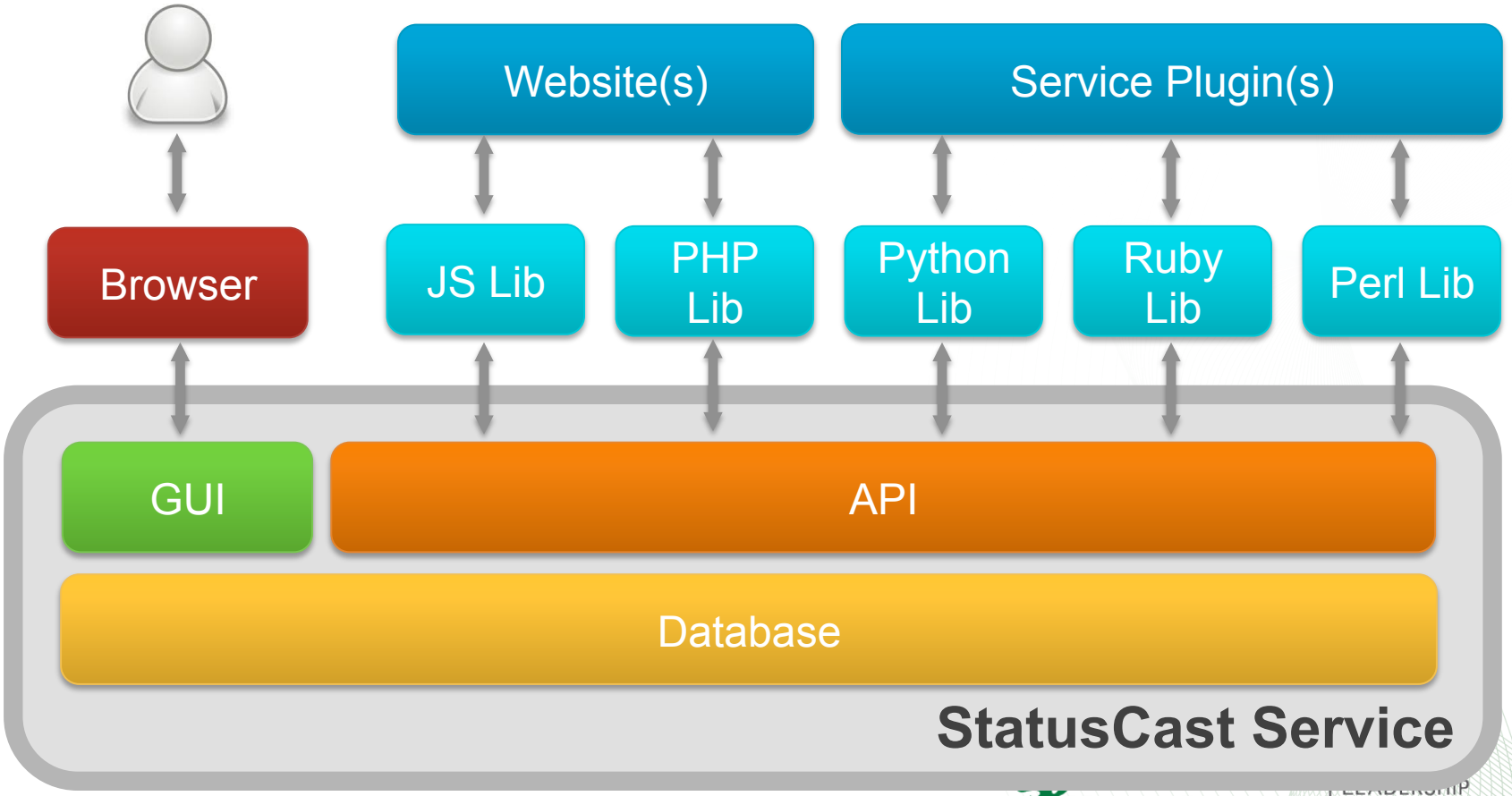
Non-SOA Shortcomings

- Many points of failure
  - Multiple scripts
  - Multiple cron jobs
  - Multiple filesystems
- Slow due to file syncing
- Hard to document and maintain
- Each script must validate inputs independently
- Schema changes nearly impossible

Case: Redesign Using Service-oriented Principles

Redesign: “StatusCast” Service

- Distinct service boundary
- Autonomous application
- Explicit, stable API



API Over HTTPS / REST

- Limit actions to “SCRUD”
- (120) exposed URLs namespaced into (3) distinct sets:
- https://<base>/gui/<resource>/<action>
  - Graphical interface via browser
  - Restrict to certain domains. 2-factor authentication.
- https://<base>/api/<version>/secure/<resource>/<action>
  - “Destructive” API calls (e.g. create, update, destroy records).
  - Restrict to certain domains. API token (1-factor) authentication.
- https://<base>/api/<version>/open/<resource>/<action>
  - “Non-destructive” API calls (e.g. show, search records).
  - Unrestricted on domains. API token (1-factor) authentication.

SOA Benefits

- Fewer points of failure
- Centralized input validation
- Centralized logging
- Schema changes need not propagate to API layer
- Decentralized developer effort and communication
- More maintainable

Service Plugins

Service Plugins

- Not every utility can be fully SOA compliant service
- Service plugins are small and make use of one or more services
- Can be written in Python, Perl, Ruby, Javascript, or PHP using provided libraries
- Service registry to store all plugin metadata for reference

Simple APIs

- Include the StatusCast lib for language (x):

```
$ irb
>> load “<path>/statuscast.rb”
>> include Statuscast
```

- Make API calls with API token:

```
>> sys = statuscast_show( ‘systems’, 1, ‘ekrjDKJS8kfhskfhje’ )
```

- Get payload, check <hash>[‘code’]. Data is in <hash>[‘body’]:

```
>> sys
{ ”code” => 200,
  ”message” => ”OK”,
     ”body” => {
         ”system” => {
               ”id” => 1,
            ”name” => ”titan”,  ...
```

Conclusions

Conclusion

- Service boundaries are explicit
- Services are autonomous
- Services share contract, not class
- Service-orientation offers substantial benefits once applications reach certain size, scope
  - More maintainable
  - More flexible
  - More reliable
  - Better logging
  - Better schema validation

Questions?