ESUG 2019

Evolving ALLSTOCKER

Agile increments with Pharo Smalltalk

SORABITO

Masashi Umezawa & Kazunori Ueda

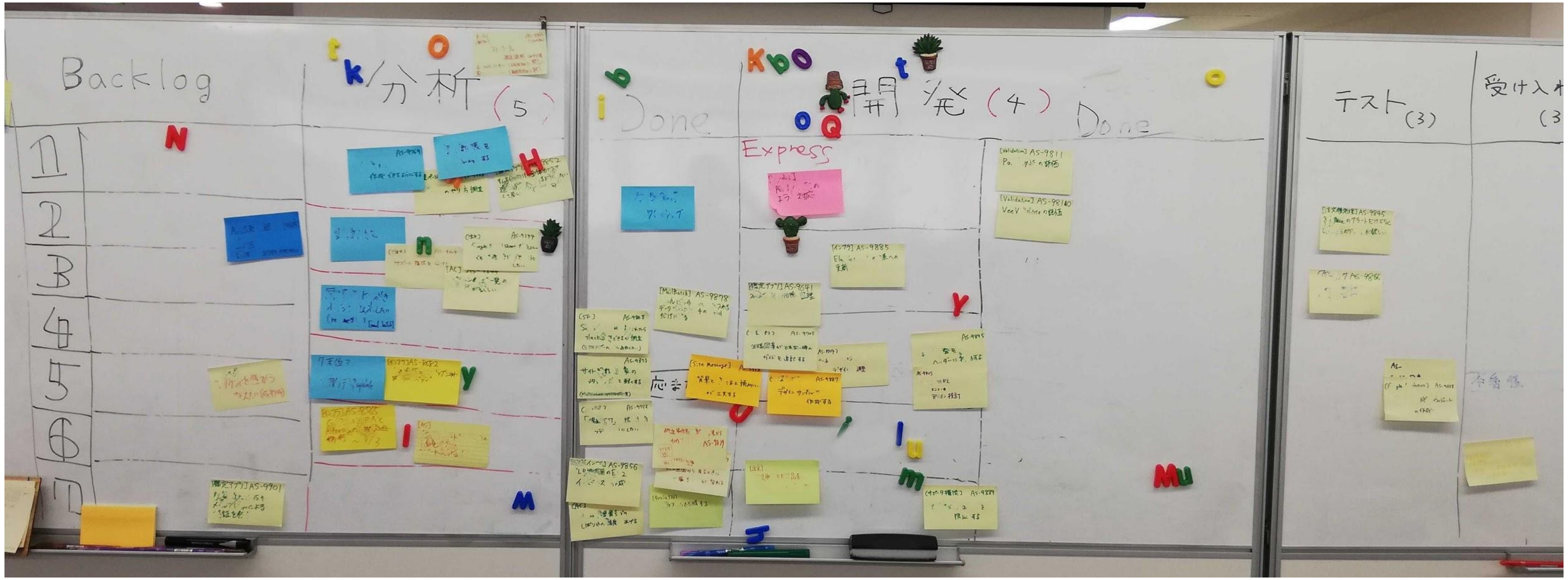
What is ALLSTOCKER?

- Online platform for trading used construction equipment and machinery
  - Marketplace
  - Real-time Bid Auction
- Over 4000 worldwide buyers
- Over 400 machines/month listed on the site
- Most systems are built with Pharo Smalltalk

Our Development Process - KANBAN

- Swarm to make the flow smoother!

Backlog → Analysis → Development → Test → Deploy



2015/02- **In the Beginning**

- First prototype was made in two weeks
  - Only 4 prerequisites (Seaside, Glorp, Nagare, AWS SDK for Smalltalk)
  - User/Machine registration, photo uploader, watchlist
- After 90 releases
  - 34 prerequisites
  - 1100+ classes

Stuck?

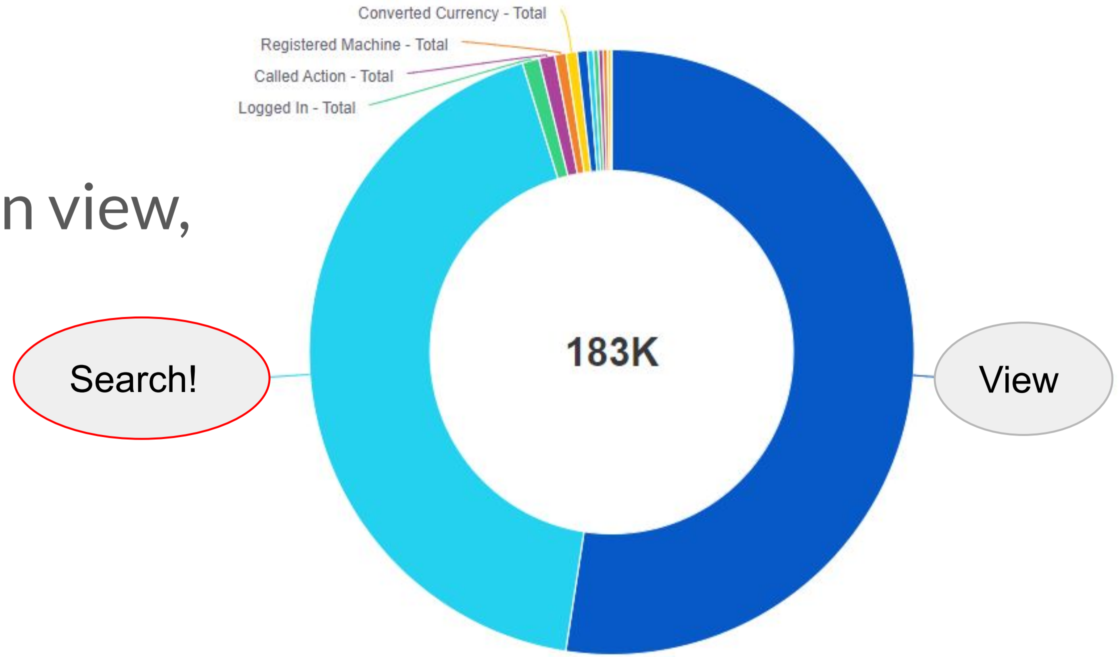
Our policy - Smalltalk as a Hub

- We take polyglot microservices approach
  - Programming languages
    - Smalltalk, JavaScript, Ruby, Lua, Groovy, Erlang, Python
  - Databases
    - PostgreSQL, Redis, Neo4j, Tarantool, MongoDB
  - External APIs
    - Elasticsearch, SendGrid, Mixpanel, Fluentd
- Smalltalk is a great hub for leveraging these elements

Marketplace Search Increments

Why Search is Important?

- In our site, Search event ratio is 43+ percent
- Buyers first search, then view, and buy



2015/7 - **Full-Text Search**

- Basic full-text search
- Elasticsearch-smalltalk for multilingual full-text search
  - Different analyzers for each language
  - Search results are boosted according to the user's primary language

Elasticsearch-Smalltalk

- Building

```
prepareSearch
    esSearch := ESSearch new index: self index.
    esSearch minScore: self minScore.
    ^esSearch
```

- Searching

```
search
    es := self prepareSearch.
    es query: self buildQuery.
    ^ es searchFrom: self offset size: self limit
```

```
buildNameMatchQuery: words fieldName: fieldName ngramBoost:
boostValue
    | phraseQuery matchQuery prefixQuery |
    phraseQuery := ESMatchQuery new
                        matchPhrase;
                        field: fieldName;
                        query: words;
                        yourself.
    prefixQuery := ESPrefixQuery new
                        field: fieldName;
                        query: words;
                        yourself.
    matchQuery := ESMatchQuery new
                        field: fieldName, '__ngram';
                        query: words;
                        boost: boostValue;
                        yourself.
    ^ ESBoolQuery new
                    should: {phraseQuery. prefixQuery. matchQuery};
                    minimum_should_match: 1;
                    yourself.
```

Boosted!

2016/2 - **Advanced Search**

- Let’s support advanced search!
  - Many aggregation options
    - by category, maker, model number
- Elasticsearch was not enough to do complex aggregations
- Joining tables with Glorp was hard to maintain
- We adopted Graph database - Neo4j

Graph Model (1)

- Nodes and Relationships
  - (Machine)-[BELONGS_TO_CATEGORY]->(Category)
  - (Machine)-[IS_OF_MODEL]->(Model)
  - (Maker)-[MADE_MODEL]->(Model)
  - (Model)-[HAS_SPEC]->(Spec)
  - (Spec)-[BELONGS_TO_SPEC_CATEGORY]->(SpecCategory)

Neo4reSt

- Neo4j database client and Object wrappers
  - Introducing Neo4reSt

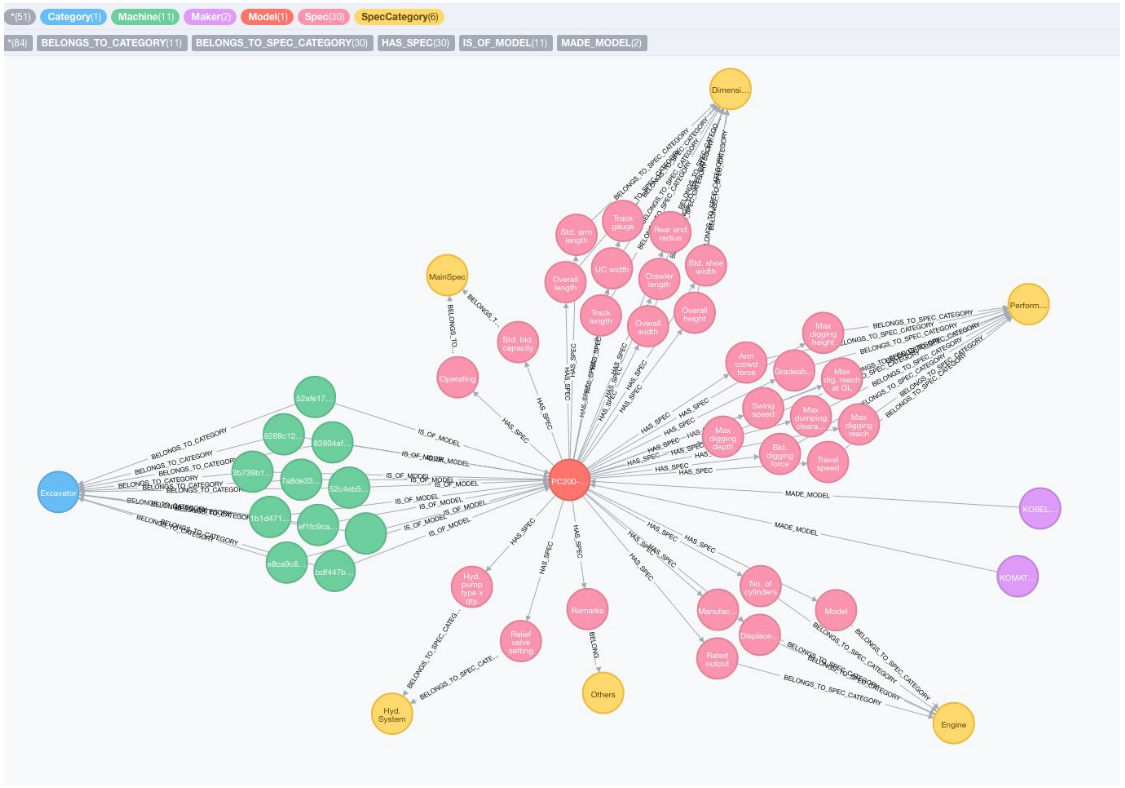
```
db := N4GraphDb new.
node1 := db createNode: {#name-> 'ume'}.
node2 := db createNode: {#name-> 'Smalltalk'}.

relation1 := node1 relateTo: node2 type: #uses properties: {#years->18}.

db initialNode relateTo: node1 type: #people
```

Graph Model (2)

- We can freely get nodes/ relationships using Cypher query language



2017/2 **Revamping Advanced Search**

- Need to generate complex queries dynamically according to various search options (spec filters)
- Hard-coded cypher queries were unmaintainable.
- SCypher was developed
  - "Manipulating Neo4j from Pharo Smalltalk" (Sample code project)

SCypher

```
user := 'user' asCypherObject.
friend := 'friend' asCypherObject.
friends := 'friends' asCypherObject.
query := CyQuery statements: {
    CyMatch of: (CyRelationship start: user end: friend type: 'FRIEND').
    CyWhere of: (CyExpression eq: (user prop: 'name') with: 'name'
asCypherParameter).
    CyWith of: (user, ((CyFuncInvocation count: friend) as: friends)).
    CyWhere of: (CyExpression gt: friends with: 10).
    (CyReturn of: user) limit: 10.
}.
query cypherString.
```

- Generated query can be executed by

  N4RestClient>>queryByCypher: queryString params: dictionary

```
'MATCH (user)-[:FRIEND]-(friend)
WHERE (user.name = $name)
WITH user, count(friend) AS friends
WHERE (friends > 10)
RETURN user LIMIT 10 '
```

Auction Increments

2016/7-9- **Beginning Realtime Auction System**

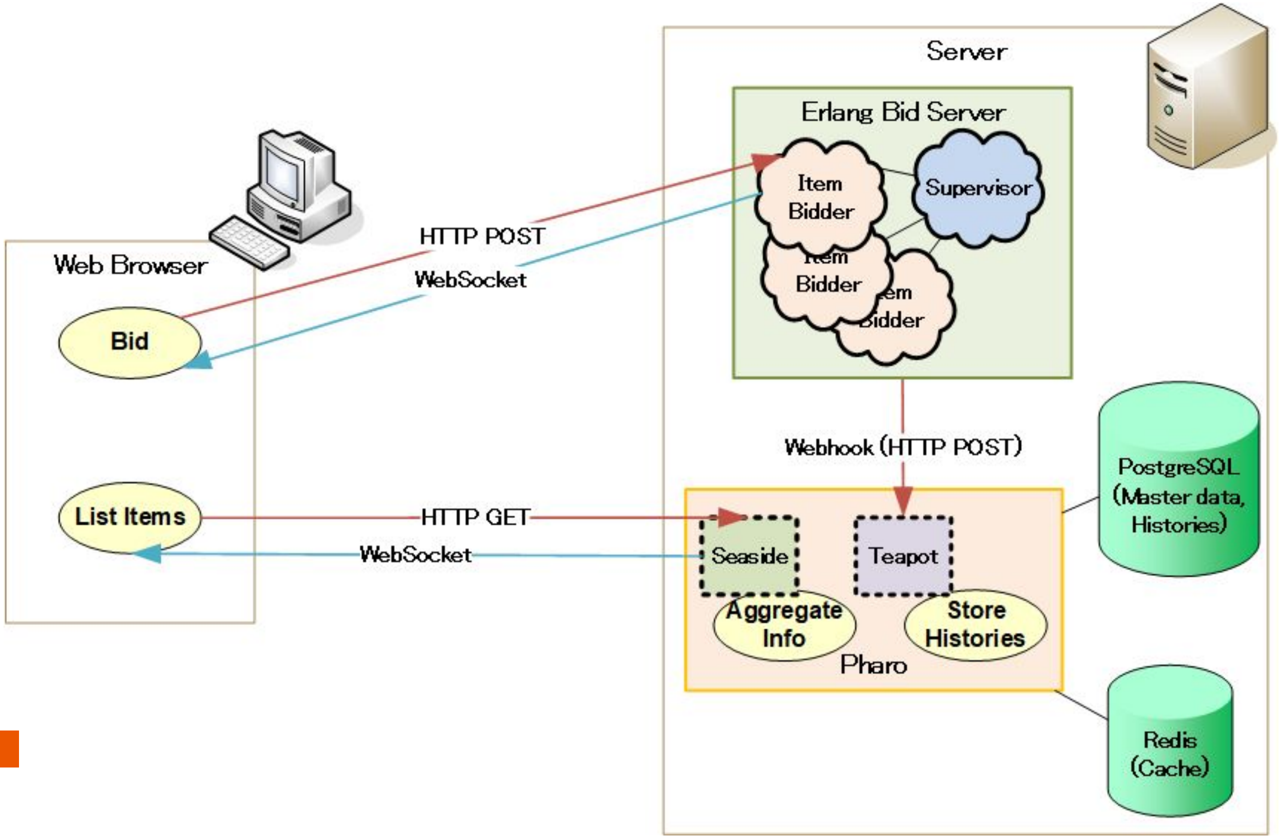
- Need a highly reactive real-time bidding auction system
    - Vue.js (presentation)
    - Pharo (business logic)
    - Erlang (bidding core)
- Web API + Ajax + WebSocket

DEMO

Original Auction System Network Architecture

- Bid requests
  - HTTP
- Notifications
  - WebSocket

But...



2018/3

Auction System Crisis

- Slow updates
- Too many connections
- Heavy load
- Difficult to log-in...

2018/4-8 **Scaling-out Auction System**

- Our strategies:

1. On-demand notifications
2. Reducing the number of connections
3. Multiple Pharo images

On-demand Notifications

- Originally each client gets periodic updates on all items
  - Via [Zinc-WebSocket]
- Each client has observing item list
  - Client gets updates only when the item values are changed
  - Via [Zinc-Server-Sent-Events]

Reducing the Number of Connections

- SSE + HTTP/2
  - Enables single TCP connection for many requests
- SSE is unidirectional and lightweight

```
server {
    listen 443 ssl http2;
    ...
}
```

Multiple Pharo images

- Let's utilize multicore CPU!
- We divided one pharo image into three
  - auction-1, auction-2 (app server)
  - webhook (interact with Erlang bid server)
- Web API + Redis pubsub for mutual communication
  - RediStick pubsub mode

RediStick

- A redis client supporting auto-reconnect
  - Reliable pubsub by pinging to itself

```
stick := RsRediStick targetUrl: 'sync://localhost'.
stick connect.
stick beSticky. "Auto reconnect when server is not accessible"
stick onError: [ :e | e pass ].
stick endpoint subscribe: #('ping') callback: [:msg | msg inspect].
"From another stick"
pubStick endpoint publish: 'ping' message: 'OK?'
```

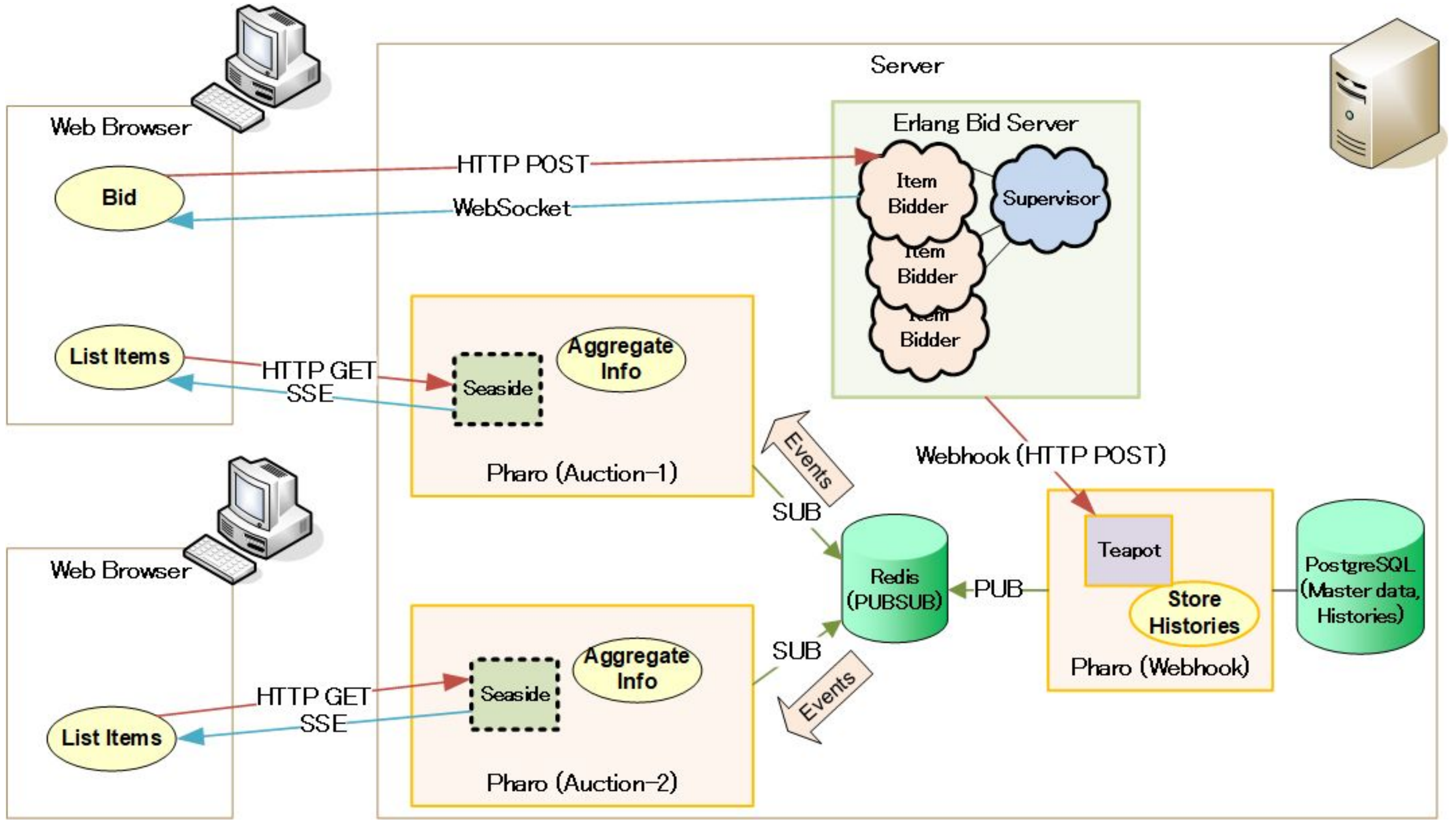
Load-balancing via nginx

- Load-balancing by cookie-based sticky session

```
upstream auction_upstream {
    hash $cookie_stocker consistent;
    server unix:/var/run/auction_1.sock;
    server unix:/var/run/auction_2.sock;
}
server {
    listen  unix:/var/run/auction_1.sock;
    location / {
      proxy_pass http://as-auction-1.default.svc.cluster.local:9000;
    }
}
server { … }
```

Revised Auction Architecture

- 3 pharo images
- PUBSUB & SSE for async notifications



DONE!!

```
$ kubectl top pods

NAME                                 CPU(cores)    MEMORY(bytes)
as-auction-1-dpl-dc699554b-c5jh4        115m          261Mi
as-auction-2-dpl-5d8bbd7b5c-qwlbp       111m          236Mi
as-auction-webhook-dpl-66f47557b-jn2dc  48m           194Mi
as-marketplace-dpl-7d898866d4-w72cc     119m          736Mi
```

Questions?

- Visit allstocker.com
- Please stay tuned for more updates!