BlueBrain Nexus

Building a knowledge graph for data driven science

March 6, 2019

Who we are

The Blue Brain Project

- Not the Human Brain Project
- Own entity under EPFL governance
- 120+ people
- 9 sections
- Supercomputer at the CSCS in Lugano

https://bluebrain.epfl.ch

https://portal.bluebrain.epfl.ch

Who we are

Two teams

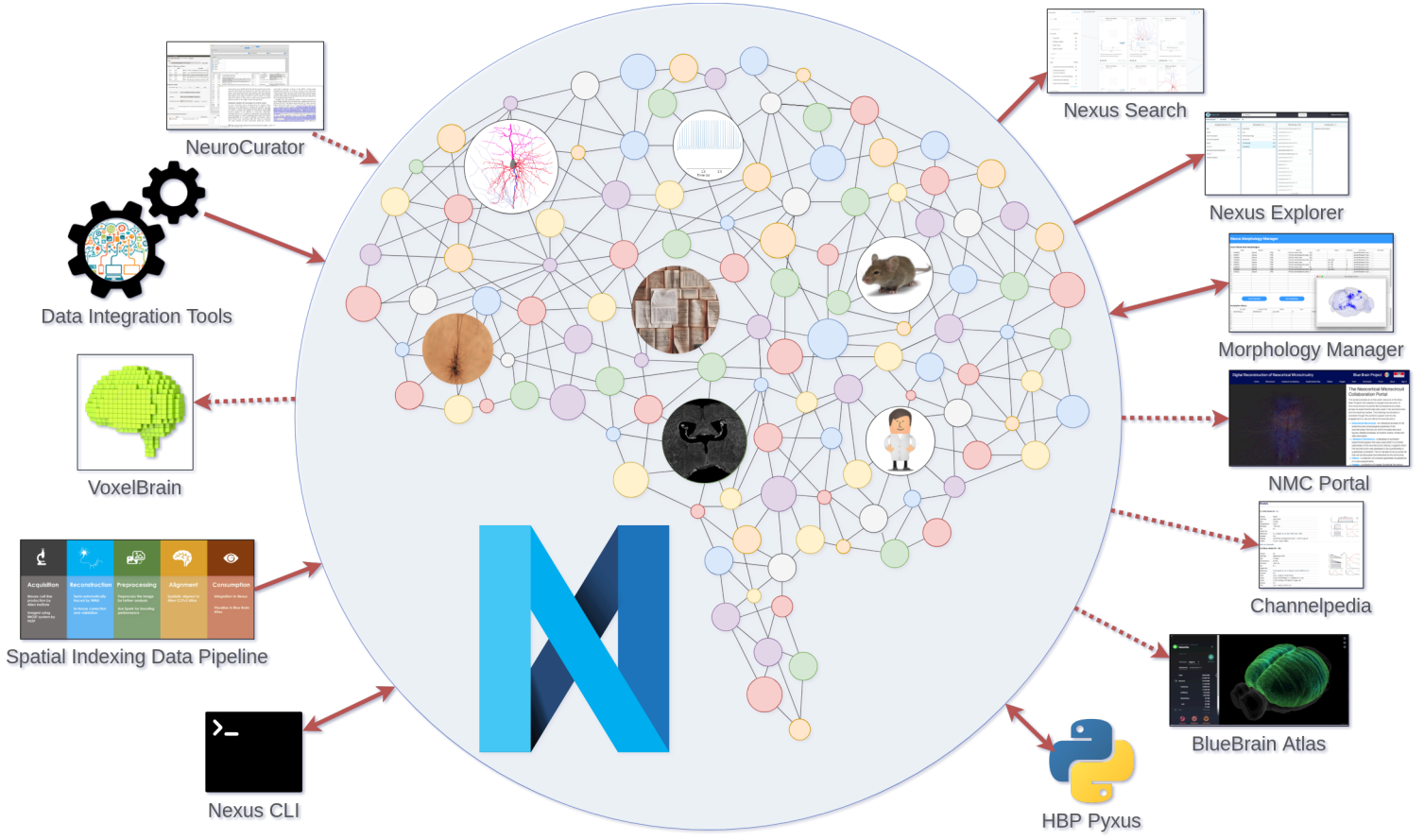
- Neuroinformatics
  - Back-end
  - Front-end
  - UI/UX
- Data & knowledge engineering
  - Modeling (vocabularies, taxonomies, ontologies, schemas)
  - Literature curation
  - Image reconstruction & atlasing

We build Nexus.

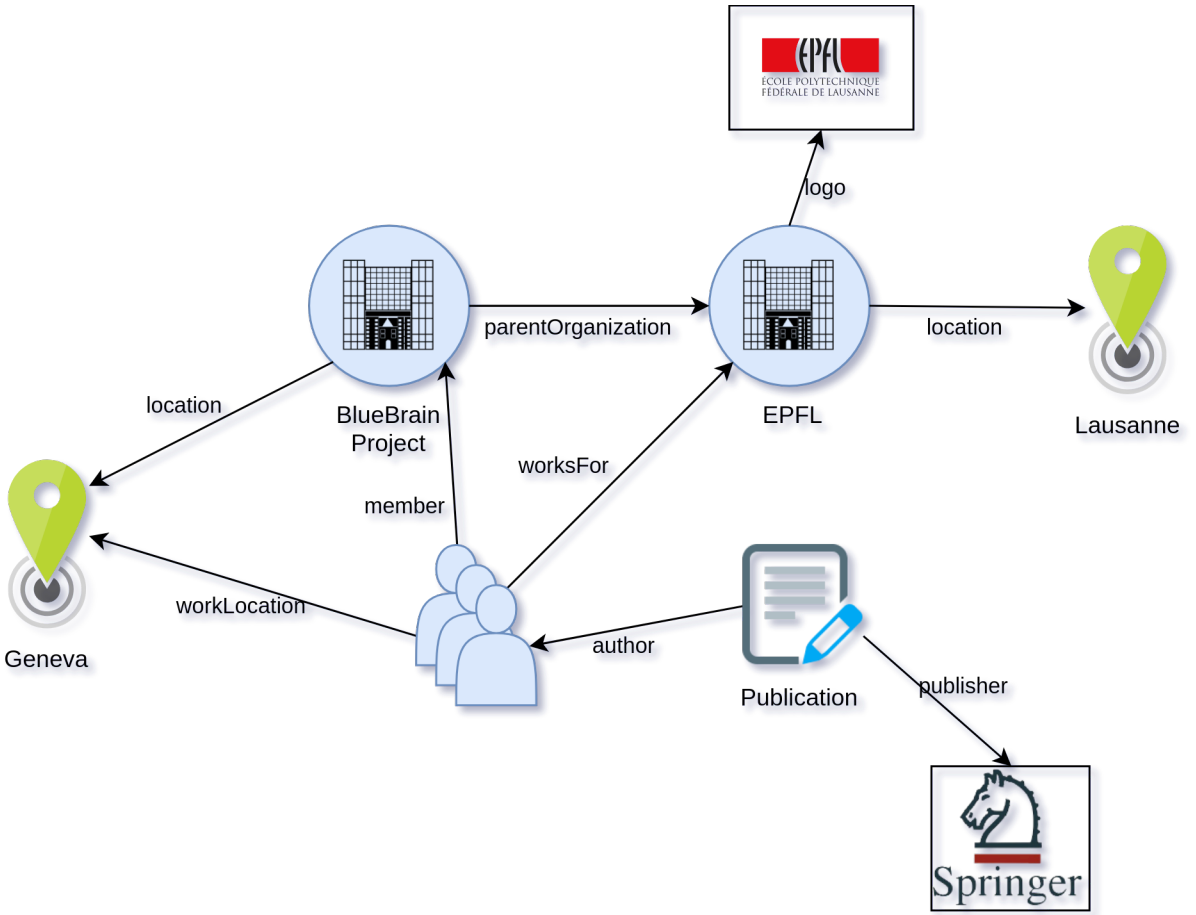
Presentation and documentation: https://bluebrain.github.io/nexus

Public instance: https://nexus-sandbox.io/web

What do we do?


NeuroCurator
Data Integration Tools
VoxelBrain
Spatial Indexing Data Pipeline
Nexus CLI
Nexus Search
Nexus Explorer
Morphology Manager
NMC Portal
Channelpedia
BlueBrain Atlas
HBP Pyxus

Why a knowledge graph?



RDF in a nutshell

RDF in a nutshell

Example from Wikipedia:

```
<rdf:RDF xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#"
         xmlns:eric="http://www.w3.org/People/EM/contact#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:mailbox rdf:resource="mailto:e.miller123@example.com"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:personalTitle>Dr.</contact:personalTitle>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.org/People/EM/contact#me">
    <rdf:type rdf:resource="http://www.w3.org/2000/10/swap/pim/contact#Person"/>
  </rdf:Description>
</rdf:RDF>
```

Other representation formats: N-Triples, Turtle, JSON-LD, etc.

N-Triples

```
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/2000/10/swap/pim/conta
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/1999/02/22-rdf-syntax-
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/2000/10/swap/pim/conta
<http://www.w3.org/People/EM/contact#me> <http://www.w3.org/2000/10/swap/pim/conta
```

JSON-LD

```
{
  "@context": {
    "contact": "http://www.w3.org/2000/10/swap/pim/contact#",
    "eric": "http://www.w3.org/People/EM/contact#",
    "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
    "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
    "xsd": "http://www.w3.org/2001/XMLSchema#"
  },
  "@id": "eric:me",
  "@type": "contact:Person",
  "contact:fullName": "Eric Miller",
  "contact:mailbox": {
    "@id": "mailto:e.miller123@example.com"
  },
  "contact:personalTitle": "Dr."
}
```

SPARQL

What are five names of people that published a paper in Springer, working for EPFL in Geneva?

SPARQL

What are five names of people that published a paper in Springer, working for EPFL in Geneva?

SPARQL

```
SELECT DISTINCT ?name
WHERE {
  ?person      a             :Person   .
  ?person      :name         ?name     .
  ?publication :author       ?person   .
  ?publication :publisher    "Springer".
  ?person      :worksFor     "EPFL"    .
  ?person      :workLocation "Geneva"  .
}
LIMIT 5
```

SHACL

```
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

schema:PersonShape
    a sh:NodeShape ;
    sh:targetClass schema:Person ;
    sh:property [
        sh:path schema:givenName ;
        sh:datatype xsd:string ;
        sh:name "given name" ;
    ] ;
    sh:property [
        sh:path schema:birthDate ;
        sh:lessThan schema:deathDate ;
        sh:maxCount 1 ;
    ] ;
    sh:property [
        sh:path schema:gender ;
        sh:in ( "female" "male" ) ;
    ] ;
    sh:property [
        sh:path schema:address ;
        sh:node schema:AddressShape ;
    ] .
```

In practice

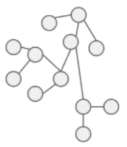
Instance

```
{
 "@context":"/contexts/neurosciencegraph/core/data/v0.1.0",
"@type":"Subject",
 "species":{
    "@id":"obo:NCBITaxon_10116"
    "label":"Rattus Norvegicus"
 }
}
```

isDefinedBy

imports

Ontology
(NCBI Taxonomy)



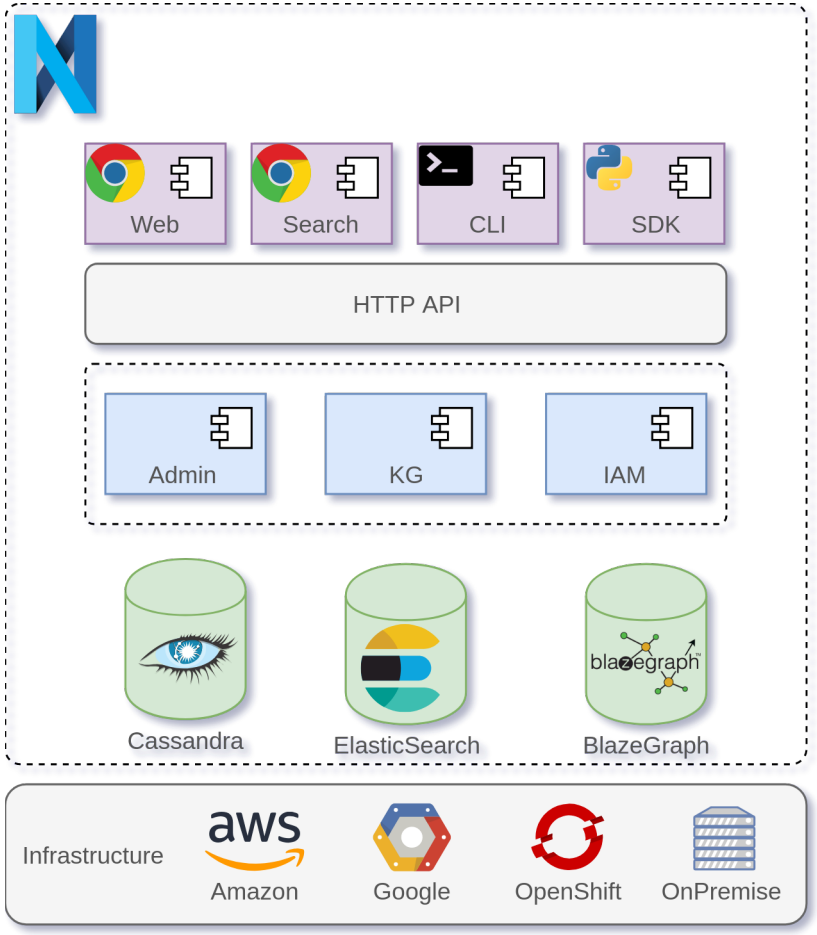
```
obo:NCBITaxon_10116 rdf:type owl:Class ;
            rdfs:label "Rattus norvegicus"@en ;
           rdfs:subClassOf obo:NCBITaxon_9989 ;
         ncbitaxon:has_rank obo:NCBITaxon_species ;
```

Schema

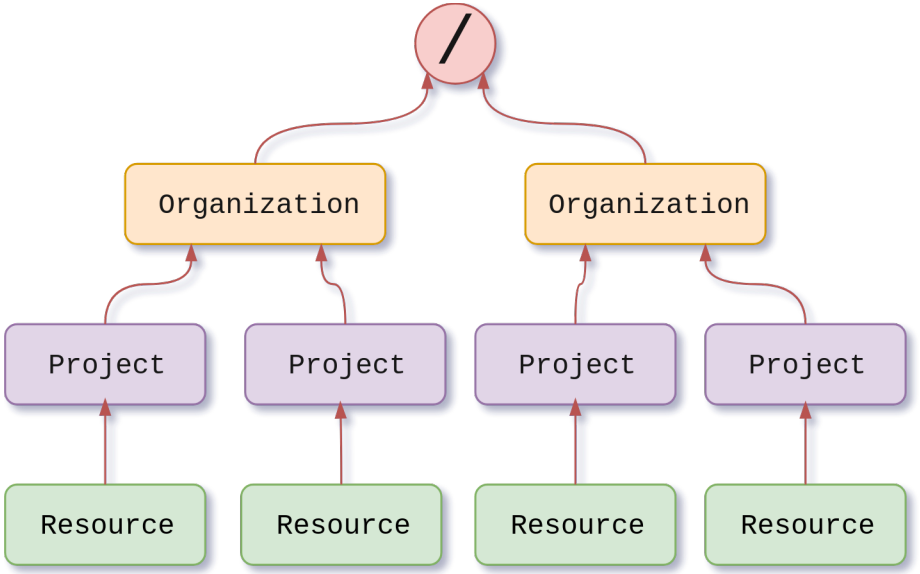
```
{
  "@context":"/contexts/neurosciencegraph/core/schema/v0.1.0",
 "imports":"/data/neurosciencegraph/ontologies/taxonomy/v0.1.0",
  "shapes":[
    {
        "@type":"sh:NodeShape",
        "targetClass":"Subject",
        "property":[{
            "path":"species",
            "node":":SpeciesOntologyTermShape",
            "minCount":1
        }
]
    },
    {
        "@id":":SpeciesOntologyTermShape",
        "@type":"sh:NodeShape",
        "property":[{
            "path":"ncbitaxon:has_rank",
            "hasValue":"obo:NCBITaxon_species",
            "minCount":1
          }
         ]
    }
  ]
}
```

Nexus services

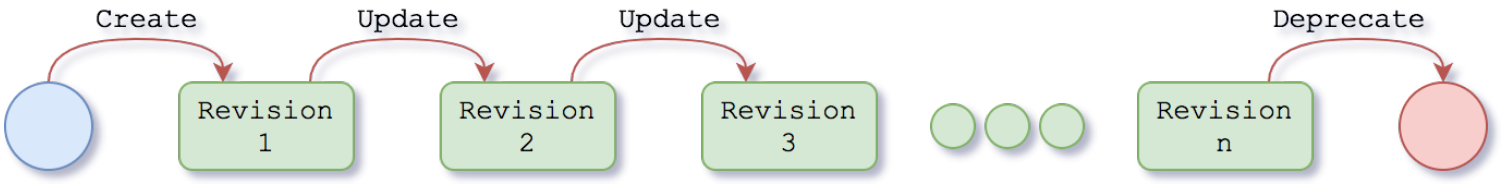
Overview


Web
Search
CLI
SDK
HTTP API
Admin
KG
IAM
Cassandra
ElasticSearch
BlazeGraph
Infrastructure
aws
Amazon
Google
OpenShift
OnPremise

Resource hierarchy



Resource life-cycle



Example with resources:

```
POST /v1/resources/{org_label}/{project_label}/{schema_id}

PUT /v1/resources/{org_label}/{project_label}/{schema_id}/{resource_id}?rev={prev}

GET /v1/resources/{org_label}/{project_label}/{schema_id}/{resource_id}

GET /v1/resources/{org_label}/{project_label}/{schema_id}/{resource_id}?rev={rev}

DELETE /v1/resources/{org_label}/{project_label}/{schema_id}/{resource_id}?rev={prev}
```

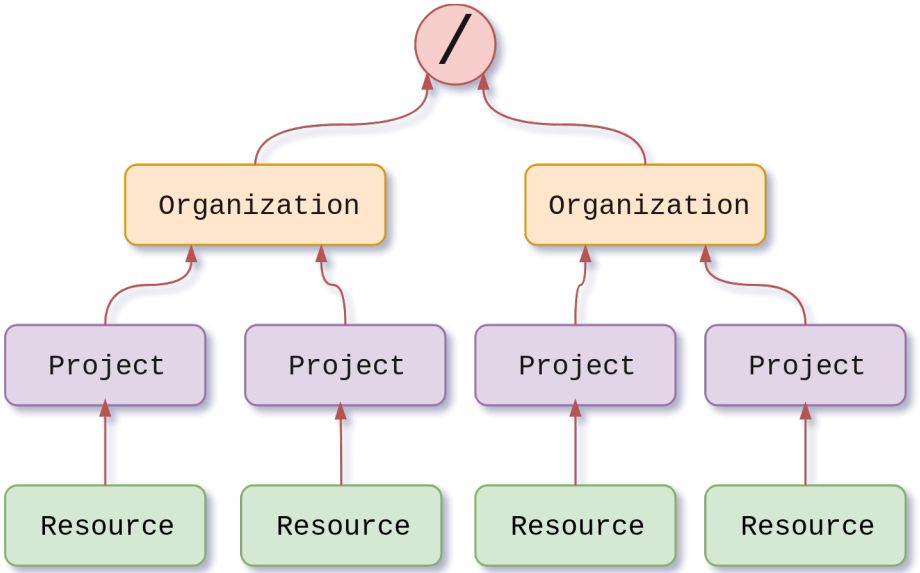
Identity & access management

- Authentication providers (OpenID Connect)
  - Realms
- Identities
  - Users
  - Groups
  - Authenticated
  - Anonymous
- Permissions
  - Arbitrary strings, e.g. `projects/create` or `schemas/write`
- ACLs
  - Resource path -> (Identity, Permissions)

```
case class AccessControlLists(value: Map[Path, AccessControlList])

case class AccessControlList(value: Map[Identity, Set[Permission]])
```

Admin

- Organizations
- Projects
  - Base
  - Vocabulary
  - API mappings (prefixes)



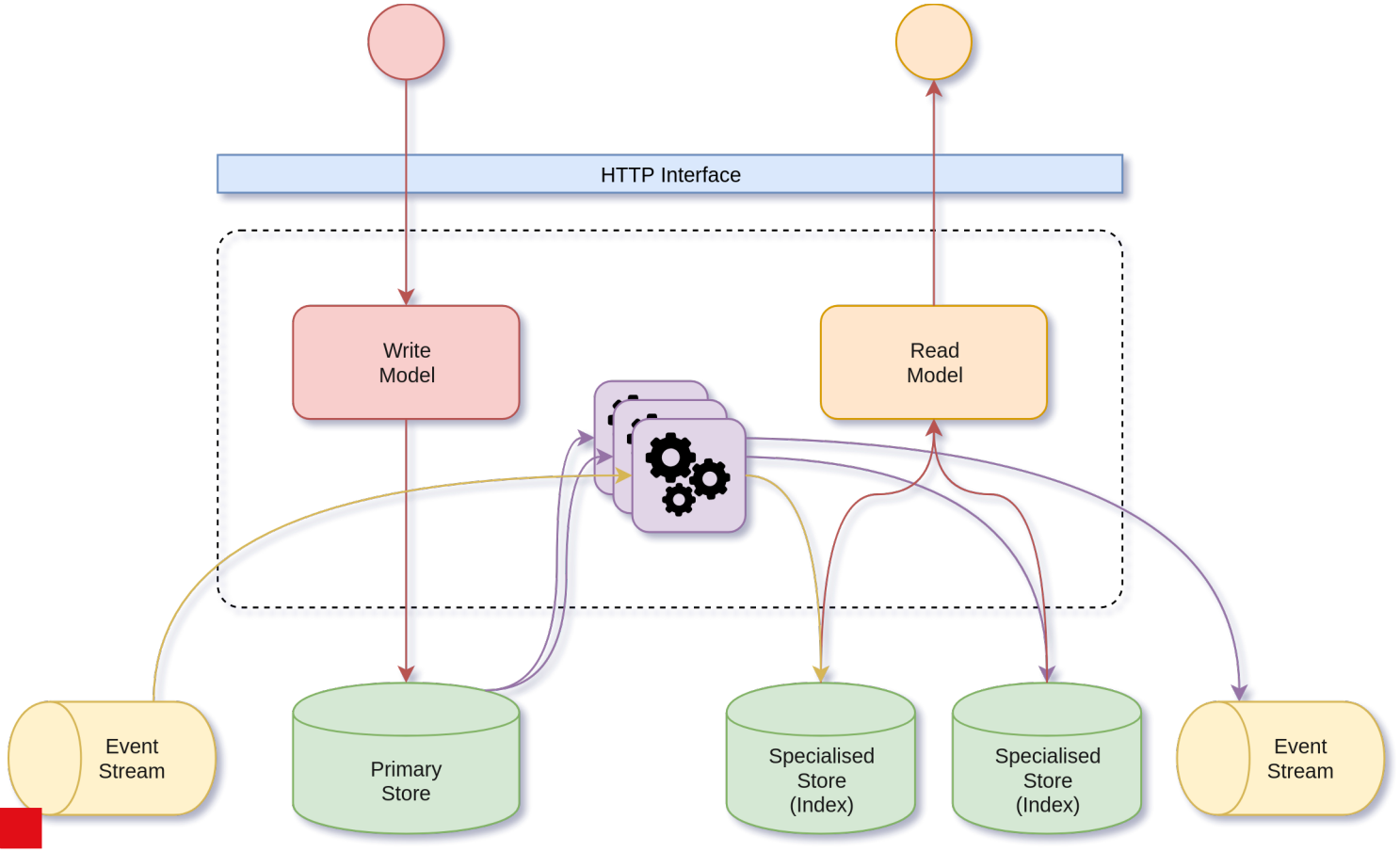
Knowledge graph

- Schemas
- Resolvers
- Views
  - Elasticsearch
  - SPARQL
- Files (binary attachments)
- Resources

Nexus components

Event sourcing & CQRS

https://github.com/BlueBrain/nexus-sourcing



Example: State

```
sealed trait State extends Product with Serializable

case object Initial extends State

final case class Current(
      id: Id[ProjectRef],
      rev: Long,
      types: Set[AbsoluteIri],
      deprecated: Boolean,
      tags: Map[String, Long],
      created: Instant,
      updated: Instant,
      createdBy: Subject,
      updatedBy: Subject,
      schema: Ref,
      source: Json
  ) extends State
```

Example: Command

```
sealed trait Command extends Product with Serializable {
  def id: Id[ProjectRef]
  def rev: Long
  def instant: Instant
  def subject: Subject
}
```

```
final case class Create(
    id: Id[ProjectRef],
    schema: Ref,
    types: Set[AbsoluteIri],
    source: Json,
    instant: Instant,
    subject: Subject
) extends Command {
  val rev: Long = 0L
}
```

Example: Event

```
sealed trait Event extends Product with Serializable {
  def id: Id[ProjectRef]
  def rev: Long
  def instant: Instant
  def subject: Subject
}
```

```
final case class Created(
    id: Id[ProjectRef],
    schema: Ref,
    types: Set[AbsoluteIri],
    source: Json,
    instant: Instant,
    subject: Subject
) extends Event {
  val rev: Long = 1L
}
```

Example: Evaluation

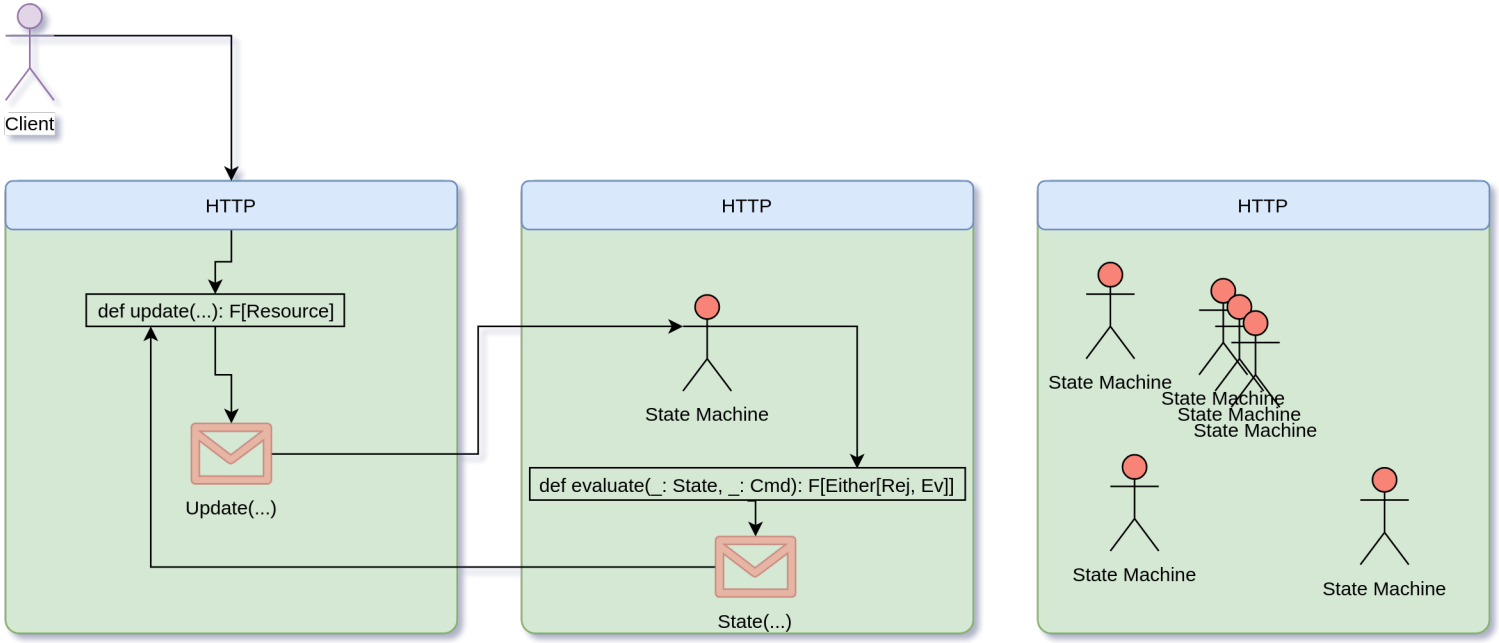
```
  def eval(state: State, cmd: Command): Either[Rejection, Event] = cmd match {
      case cmd: Create => create(state, cmd)
      // Update, Deprecate, Tag, ...
    }

  def create(state: State, c: Create): Either[Rejection, Created] =
    state match {
      case Initial => Right(Created(c.id, c.schema, c.types, c.source, c.instant,
      case _       => Left(ResourceAlreadyExists(c.id.ref))
    }
```

```
class Repo[F[_]: Monad](agg: Agg[F], clock: Clock, toIdentifier: ResId => String)

  def create(id: ResId, schema: Ref, types: Set[AbsoluteIri],
             source: Json, instant: Instant = clock.instant)(
             implicit subject: Subject): EitherT[F, Rejection, Resource] =
    evaluate(id, Create(id, schema, types, source, instant, subject))
}
```

Example: State machine

```
def next(state: State, ev: Event): State =
  (state, ev) match {
    case (Initial, e @ Created(id, schema, types, value, tm, ident)) =>
      Current(id, e.rev, types, false,
              Map.empty, None, tm, tm,
              ident, ident, schema, value)
    case (c: Current, Updated(_, rev, types, value, tm, ident)) =>
      c.copy(rev = rev, types = types,
             source = value, updated = tm,
             updatedBy = ident)
    // Deprecated, TagAdded, ...
  }
```

Example: State machine



Example: Aggregate

```
type Agg[F[_]] = Aggregate[F, String, Event, State, Command, Rejection]
```

```
def aggregate[F[_]: Effect: Timer](implicit as: ActorSystem,
      mt: ActorMaterializer, sourcing: SourcingConfig, F: Monad[F]): F[Agg[F]] =
  AkkaAggregate.sharded[F](
        name = "resources",
        Initial,
        next,
        (state, cmd) => F.pure(eval(state, cmd)),
        sourcing.passivationStrategy,
        Retry(sourcing.retry.retryStrategy),
        sourcing.akkaSourcingConfig,
        sourcing.shards
      )
```

Aggregate

```
trait Aggregate[F[_], Id, Event, State, Command, Rejection]
    extends StatefulEventLog[F, Id, Event, State] {

    def evaluate(id: Id, command: Command): F[Either[Rejection, (State, Event)]]
}
```

Using Akka cluster sharding, we construct it like this:

```
object AkkaAggregate {

  def shardedF[F[_]: Effect, Event, State, Command, Rejection](...) = {
    val shardExtractor = { case msg: Msg =>
      math.abs(msg.id.hashCode) % shards toString
    }
    val entityExtractor = { case msg: Msg => (msg.id, msg) }
    val props = AggregateActor.shardedProps(name, initialState, next,
                                            evaluate, passivationStrategy, config)
    val ref = ClusterSharding(as).start(name, props, settings,
                                        entityExtractor, shardExtractor)
    // route all messages through the sharding coordinator
    val selection = ActorRefSelection.const(ref)
    new AkkaAggregate(name, selection, retry, config)
  }
}
```

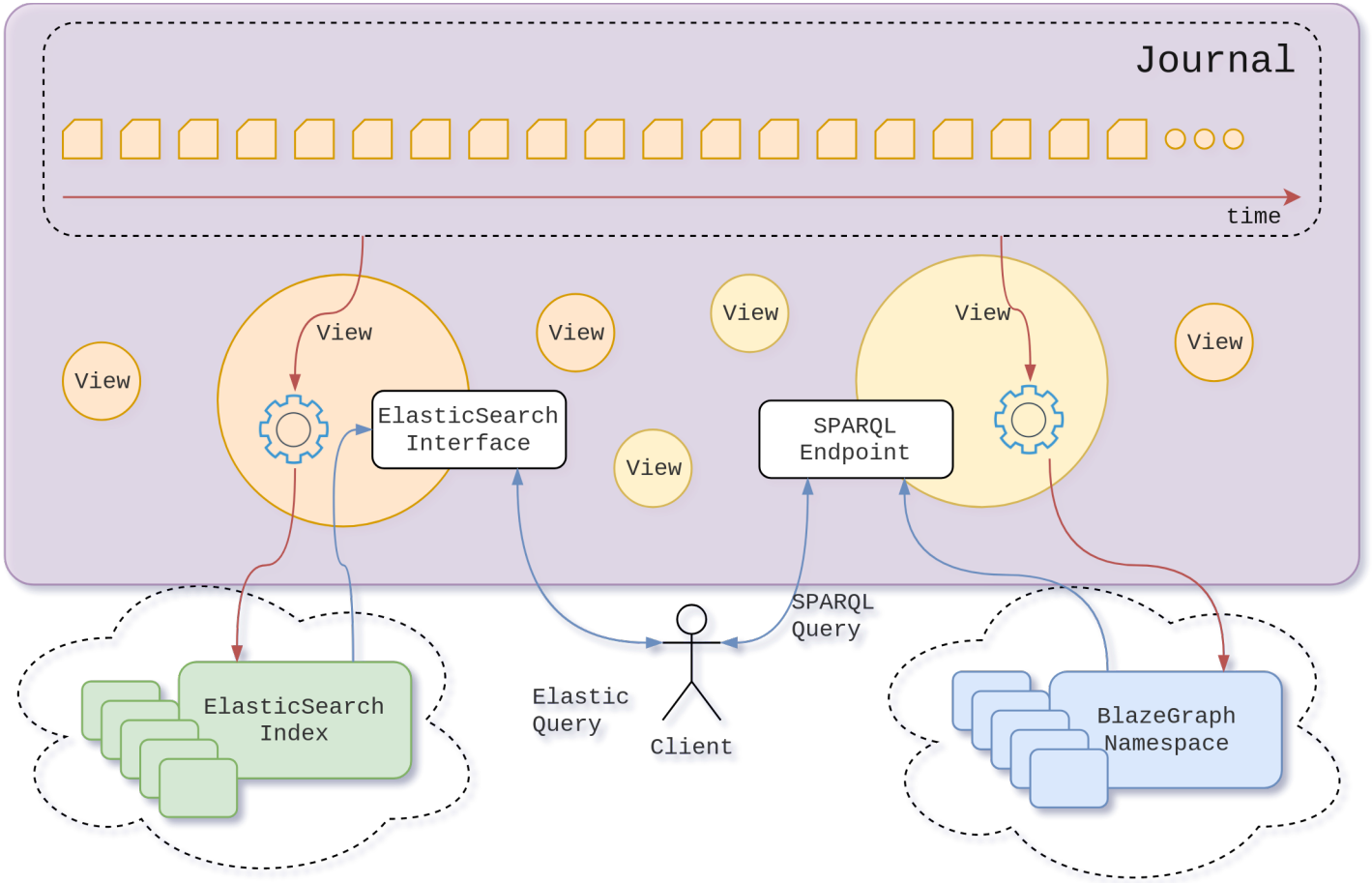
Clustered Akka aggregate

```
class AkkaAggregate[F[_]: Async, Event: ClassTag, State, Command, Rejection] (
    override val name: String,
    selection: ActorRefSelection[F],
    retry: Retry[F, Throwable],
    config: AkkaSourcingConfig,
)(implicit as: ActorSystem, mat: ActorMaterializer)
    extends Aggregate[F, String, Event, State, Command, Rejection] {

  override def evaluate(id: String, command: Command):
      F[Either[Rejection, (State, Event)]] =
    send(id, Evaluate(id, command), r => r.value)

  private def send[Reply, A](id: String, msg: Msg, f: Reply => A)
      (implicit Reply: ClassTag[Reply]): F[A] =
    selection(name, id).flatMap { ref =>
      val future = IO(ref ? msg)
      val fa     = IO.fromFuture(future).to[F]
      fa.flatMap[A] {
          case Reply(value) => F.pure(f(value))
          case e            => F.raiseError(e)
        }
        .retry
    }
}
```

Persistent aggregate actor

```
class AggregateActor[F[_]: Effect, Event, State, Command, Rejection](
    name: String,
    initialState: State,
    next: (State, Event) => State,
    evaluate: (State, Command) => F[Either[Rejection, Event]],
    passivationStrategy: PassivationStrategy[State, Command],
    config: AkkaSourcingConfig,
) extends PersistentActor with Stash with ActorLogging {

  private def evaluateCommand(cmd: Command, test: Boolean = false): Unit = {
    val eval = for {
      _ <- IO.shift(config.commandEvaluationExecutionContext)
      r <- evaluate(state, cmd).toIO.timeout(config.commandEvaluationMaxDuration)
      _ <- IO.shift(context.dispatcher)
      _ <- IO(self ! r)
    } yield ()
    val io = eval.onError { ... } // error handling
    io.unsafeRunAsyncAndForget()
  }
}
```

Indexing


Journal
time
View
View
View
View
View
View
View
ElasticSearch
Interface
SPARQL
Endpoint
SPARQL
Query
Elastic
Query
Client
ElasticSearch
Index
BlazeGraph
Namespace

Indexing

```
case class IndexerConfig[F[_], Event, MappedEvt, Err, O <: OffsetStorage](
    tag: String,
    pluginId: String,
    name: String,
    mapping: Event => F[Option[MappedEvt]],
    index: List[MappedEvt] => F[Unit],
    init: F[Unit],
    batch: Int,
    batchTo: FiniteDuration,
    retry: Retry[F, Err],
    storage: O)
```

```
trait StreamByTag[F[_], A] {

  def fetchInit: F[A]

  def source(init: A): Source[A, _]
}
```

Indexing

```
class VolatileStreamByTag[F[_]: Effect, Event, MappedEvt, Err](
      config: IndexerConfig[F, Event, MappedEvt, Err, Volatile])
      extends StreamByTag[F, Offset] {

  def batchedSource(initialOffset: Offset):
        Source[(Offset, List[IdentifiedEvent]), NotUsed] = {
      val eventsByTag =
        PersistenceQuery(as)
        .readJournalFor[EventsByTagQuery](config.pluginId)
        .eventsByTag(tag, initialOffset)
      // ... casting, mapping, batching
      eventsBatched
    }

  def fetchInit: F[Offset] =
    if (config.storage.restart)
      config.init.retry *> F.pure(NoOffset)
    else
      config.init.retry.flatMap(_ => projection.fetchLatestOffset.retry)
}
```

Indexing

```
class VolatileStreamByTag[F[_]: Effect, Event, MappedEvt, Err](
      config: IndexerConfig[F, Event, MappedEvt, Err, Volatile])
      extends StreamByTag[F, Offset] {

  def source(initialOffset: Offset): Source[Offset, NotUsed] = {

    val eventsIndexed = batchedSource(initialOffset).mapAsync(1) {
      case (offset, events) =>
        val index =
          config.index(events.map { case (_, _, mapped) => mapped })
          .retry
          .recoverWith(recoverIndex(offset, events))
        F.toIO(index.map(_ => offset)).unsafeToFuture()
    }

    val eventsStoredProgress = eventsIndexed.mapAsync(1) { offset =>
      F.toIO(projection.storeLatestOffset(offset)
      .retry
      .map(_ => offset))
      .unsafeToFuture()
    }

    eventsStoredProgress
  }
}
```

Indexing: how it's used

- Define indexers
  - Elasticsearch
  - Triple store (Blazegraph)
  - Akka Distributed Data
- Wrap the Akka source in an coordinator actor
- Run the coordinator as an Akka Cluster Singleton or Cluster Sharded actor.

Interservice communication

We had Kafka

- Easy to integrate with Alpakka Kafka
- Drawbacks:
  - One more thing to operate (several nodes + ZooKeeper)
  - Exposing event streams to external users

Solution: server sent events

- Support included in Akka HTTP
- Plain HTTP
  - Access restriction with Nexus ACLs

SSE: producer

```
val pq = PersistenceQuery(as).readJournalFor[EventsByTagQuery](queryJournalPlugin)

def source(
    tag: String,
    offset: Offset
)(implicit enc: Encoder[Event]): Source[ServerSentEvent, NotUsed] =
  pq.eventsByTag(tag, offset)
    .flatMapConcat(eventEnvelope => Source(eventToSse(eventEnvelope).toList))
    .keepAlive(10 seconds, () => ServerSentEvent.heartbeat)

def lastEventId: Directive1[Offset] =
  optionalHeaderValueByName(`Last-Event-ID`.name)
    .map(_.map(id => `Last-Event-ID`(id)))
    .flatMap {
      case Some(header) =>
        Try[Offset](TimeBasedUUID(UUID.fromString(header.id)))
          .orElse(Try(Sequence(header.id.toLong))) match {
          case Success(value) => provide(value)
          case Failure(_)     => reject(validationRejection("Invalid header"))
        }
      case None => provide(NoOffset)
    }
```

SSE: consumer

```
class EventSource[A: Decoder] {

  def send(request: HttpRequest)(
      implicit cred: Option[AuthToken]): Future[HttpResponse] =
    http.singleRequest(addCredentials(request)).map { resp =>
      if (!resp.status.isSuccess())
        logger.warn(s"Error when performing SSE request: '${resp.status}'")
      resp
    }

  def apply(iri: AbsoluteIri, offset: Option[String])(
      implicit cred: Option[AuthToken]): Source[A, NotUsed] =
     SSESource(iri.toAkkaUri, send, offset, sseRetryDelay).flatMapConcat { sse =>
      decode[A](sse.data) match {
        case Right(ev) => Source.single(ev)
        case Left(err) =>
          logger.error(s"Failed to decode admin event '$sse'", err)
          Source.empty
      }
    }
}
```

Monitoring

We use Kamon with two reporters:

- Prometheus for metrics, plotted in Grafana
- Jaeger for tracing
- Be careful not to generate too many metrics
  - `operationName` directive creates a new histogram for every different request path
  - Start with a reasonable sampling rate

Nexus RDF

https://github.com/BlueBrain/nexus-rdf

IRI

- Superset of URI that supports unicode, used by RDF
- Custom parser implemented with Parboiled2
- Manipulation (scheme, host, port, path, segments, query, ...)
- Conversion (absolute, relative, URL, URN, CURIE, ...)

Graph

- Construction
- Manipulation
  - Nodes
  - Triples
- Serialization
  - JSON-LD with Circe and Jena
  - N-Triples
  - DOT graph format

Application: Nexus Search

https://bbp.epfl.ch/nexus/search/

Questions?