How to build
Game Server **2.0**

Peter Rybár

Requirements

- **Performance**
  - Fast responses < 0.1 s
- **Scalability**
  - Number of player sessions ~100k
- **High Availability**
  - 24/7 service
- **Fault Tolerance**
  - Minimize outages

Principles

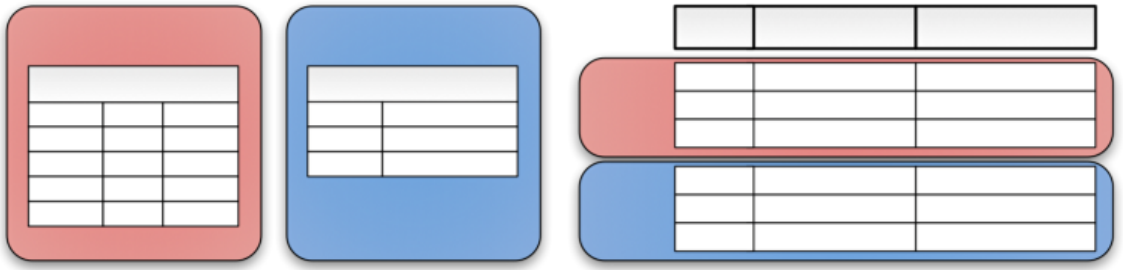
- **Linear Scalability**
  - No manual configuration!
  - Automatic **upscale, downscale**
- **IMDG** – In Memory Data Grid
  - Distributed data – **performance**
  - Replicated IM data – **Fault Tolerance** (FT)
- **Cluster**
  - **High Availability** (HA)
  - Zones based on application node roles

Data

- **Configuration** data
  - Games, Campaigns… – **small volumes**
  - Central data, cacheable – massive **reading**
- **Players** data
  - Wallets, Running game sessions – **small volumes**
  - Central data, cacheable – massive **reading** and **writing**
- **Operation** data
  - Application specific, easy to change – **runtime data**
  - Optimized for performance (in memory) – massive **reading** and **writing**
- **Historical** data
  - Game rounds, sessions – **big volumes**
  - Time based stream of events – massive **writing**
  - Invoicing, Audit, Replay games
  - Important for production, not needed for demo / test

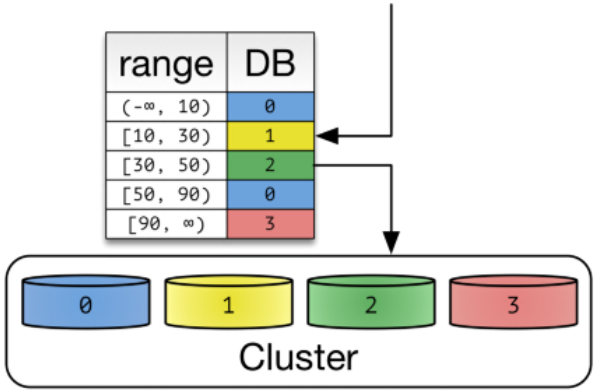
Data management

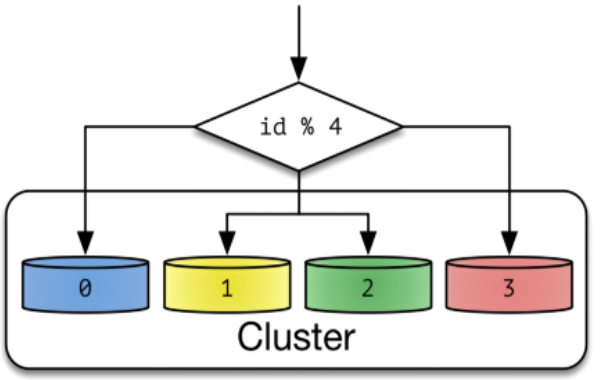
- **Sharding** – Horizontal partitioning
  - Way how to **scale Big Data volumes**
  - Way how to **scale IO** and **DB transaction**
  - Storing rows of a same table in multiple databases
  - **Use proper database for specific data category**
  - Some databases are natively sharded – Cassandra, HBase, HDFS, and MongoDB

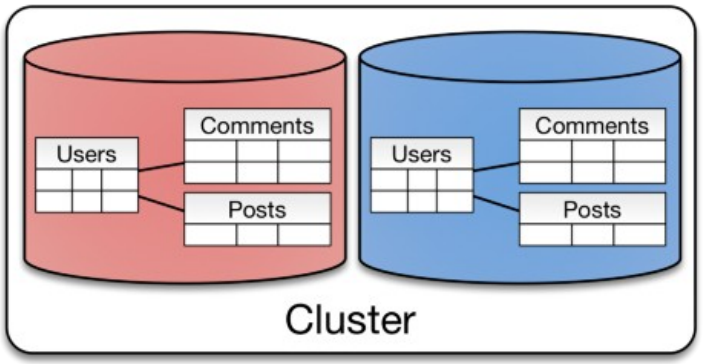


Data sharding

- **Algorithmic Sharding**
  - Uniform distribution of data
  - Determine a given partition's database without any help
- **Dynamic Sharding**
  - Nonuniform distribution of data
  - Clients need to consult the locator service first
- **Entity Groups**
  - Queries within a single physical shard (RDBM) are efficient.
  - Stronger consistency (ACID) semantics can be achieved within a shard.





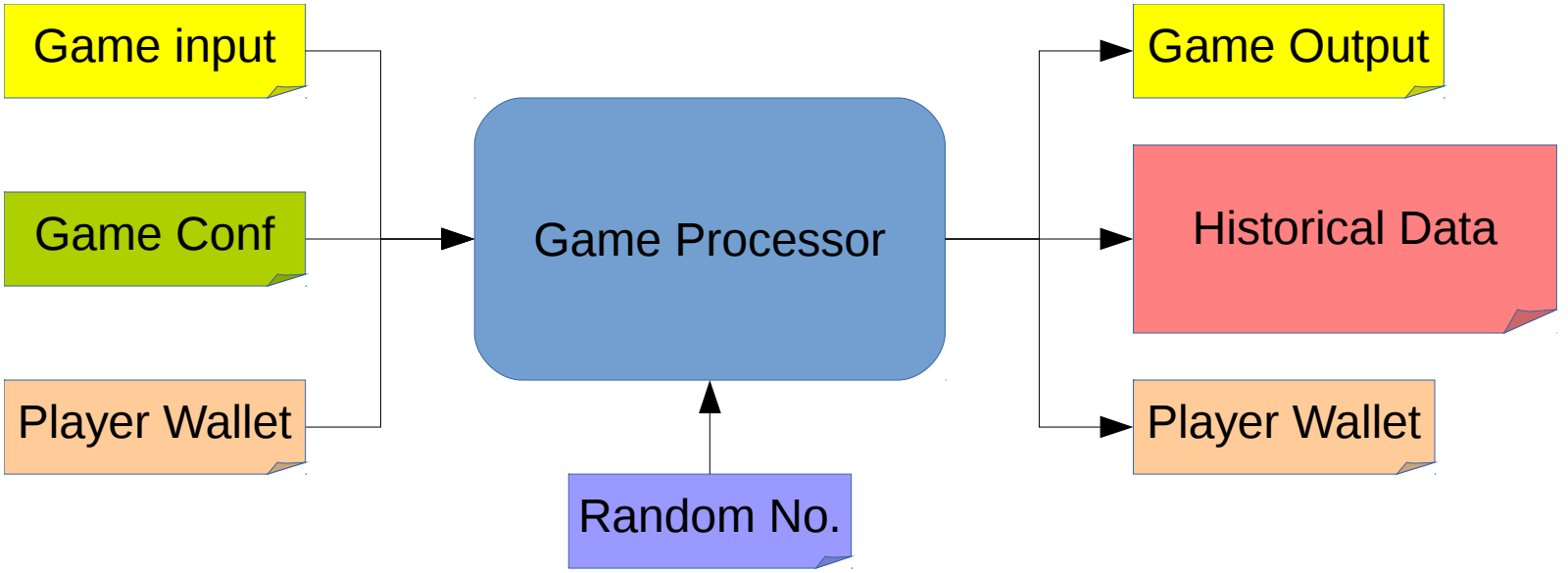


Game data

- **Players data**
  - **Entity groups** sharding
  - Sharding by **player ID**
  - Partitioning by player, game session
    - scalability of sharding while retaining most of its flexibility
- **Historical data**
  - Algorithmic sharding
  - Dynamic sharding
- **Polyglot Persistence**
  - using multiple data storage **based upon the way data is being used** by application

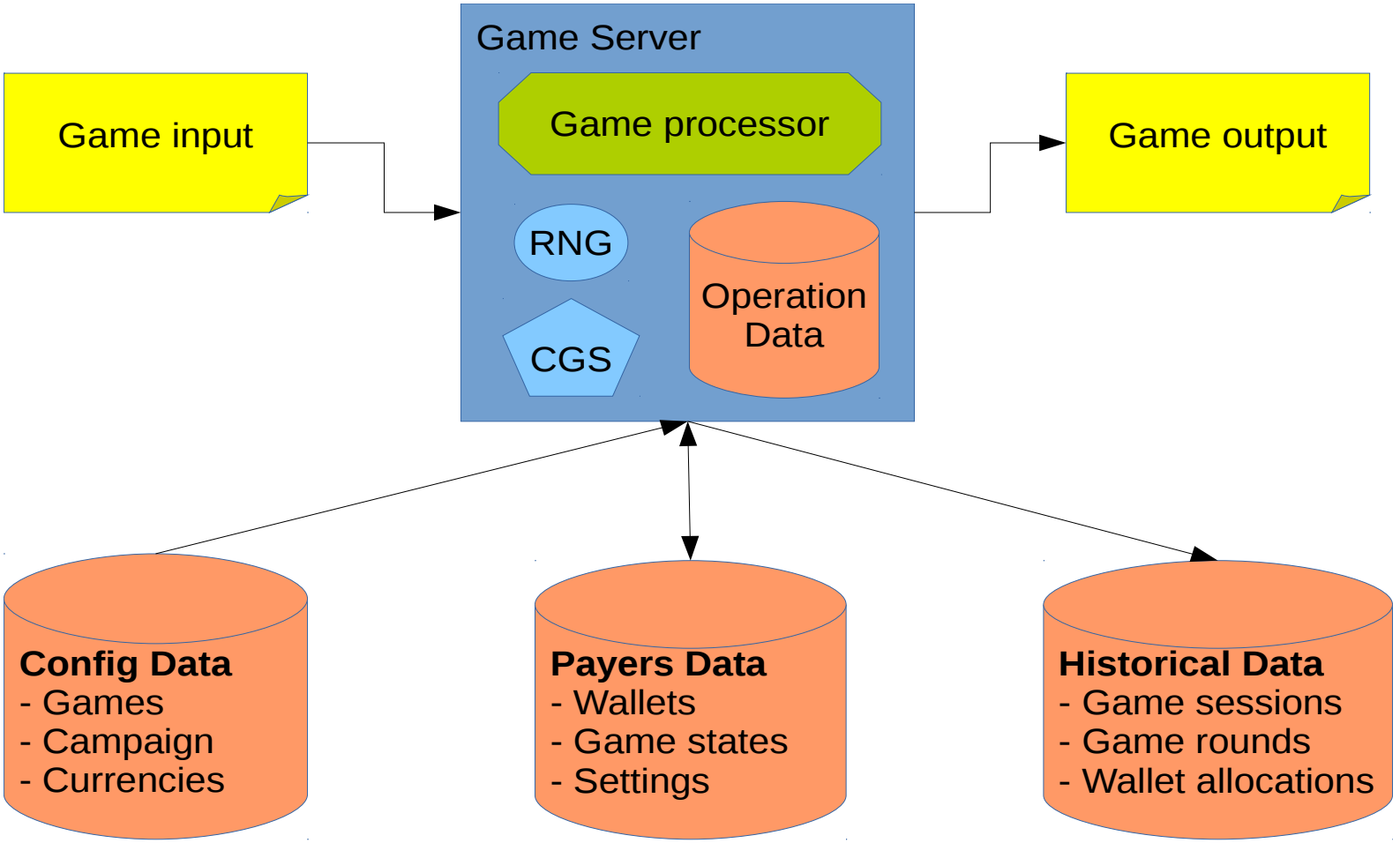
Game processing – IO

- Separate game processor code from data
- Separate data persistence from game logic

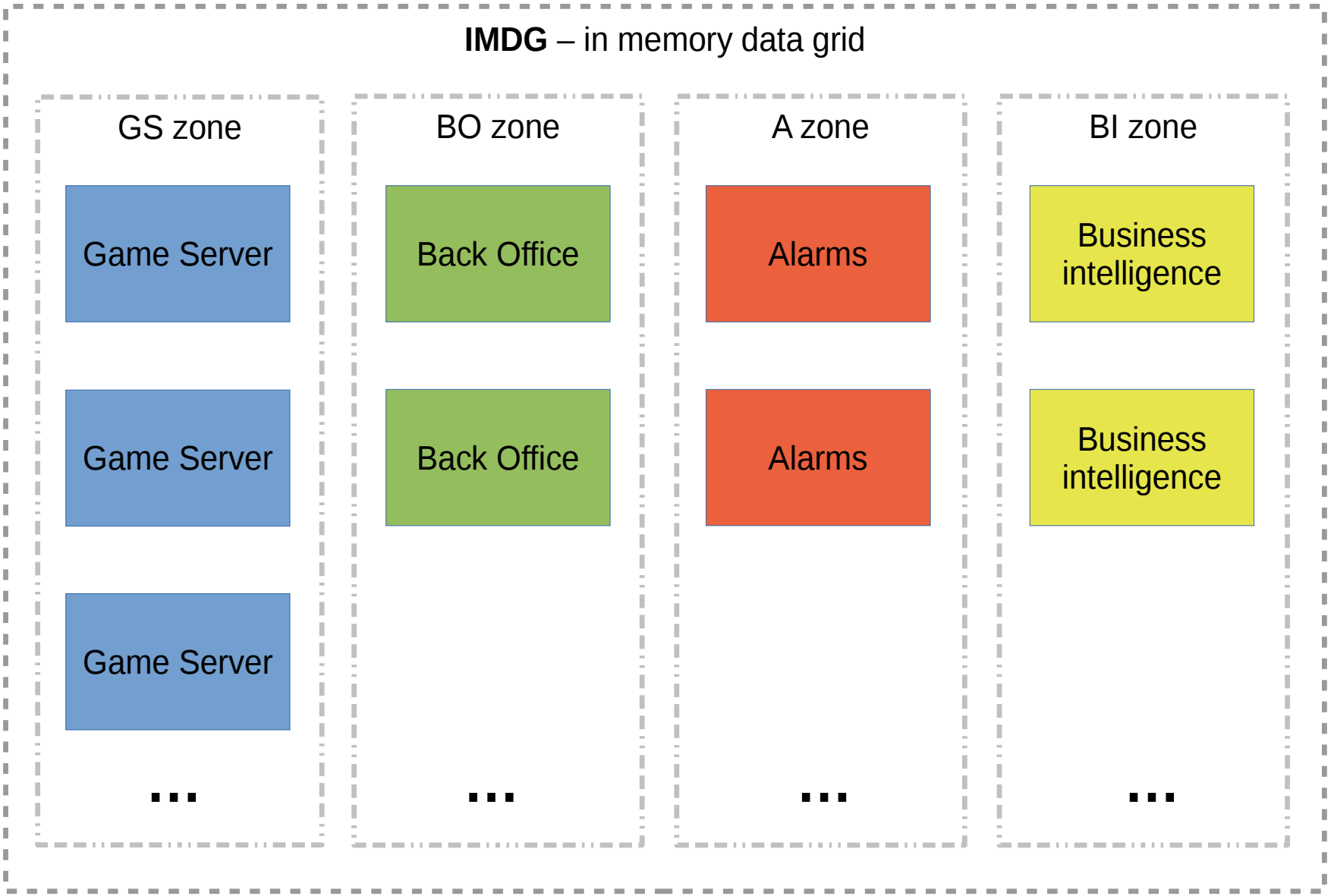


Data persistence

Polyglot Persistence – using multiple data storage (**RDBMS**, **NoSql**, **IMDG**) based upon the way data is being used by applications



Architecture – Scaling, FT, HA



Communication – GS ↔ Casino

- **Game Server ↔ Casino platform**
  - **Money transaction based** protocol – **not game based!**
    - Money transactions are only mandatory to operate
  - **Game specific meta-data** – protocol extensions
    - Extensions for different types of games – even **social gaming**
  - Casino interaction (IO) strategy – **scale casino IO** !!!
    - Session Wallet – IO once per game session
    - Deep Wallet – IO once per game round
    - **Hybrid Wallet** – IO based on casino performance
      - Different strategies for casinos, game types, wallet value, …
      - Deep Wallet, Session Wallet – only corner cases
  - **One protocol** implementation – **not two** (DW, SW)
    - Hybrid Wallet – **Simpler casino platform integration**

Technologies

- **Java 8** – streams, lambdas, datetime types, …
- **IMDG** – Apache Ignite
  - Clustering – auto cluster discovery
  - Cache – decrease IO, increase performance
  - Transactions, concurrency
  - Memory data replication – **FT, HA**
- **PostgreSQL** – best open source RDBMS, JSON support
- **Jetty** Web server – fast, embedded, lightweight
- **DbUtils, Datanucleus** – RDMBS persistence
- **FasterXML** – serializers to JSON, MsgPack, XML
- **Signal / Slot** pattern – QT like, modularity, reusability
- **React** – web user interface

…

Project roadmap

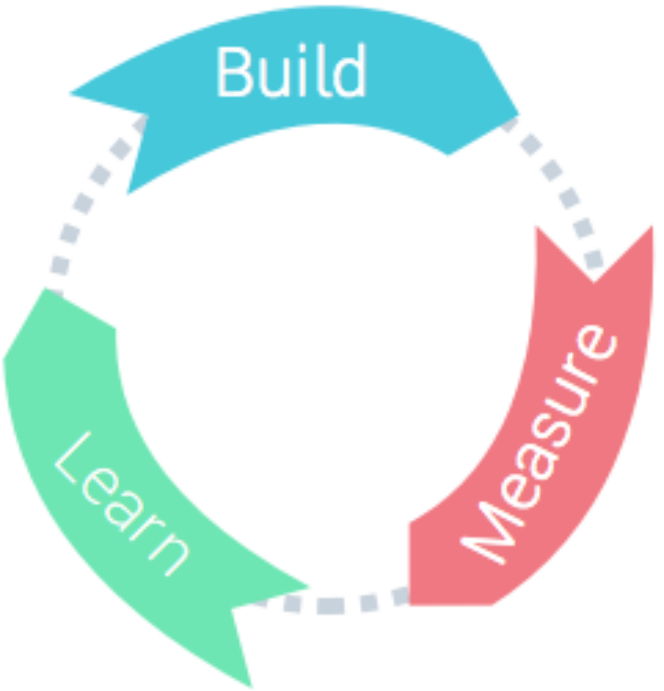
- **Summarize requirements**
  - Business and technical requirements
- **Technical analysis**
  - GS core functionality, determine modules
- **Technical design – Architecture**
  - GS core, GS modules and protocols
  - Crucial part – **Art** of SW architecture
- **Prototyping** of GS 2.0
  - Performance tests, Storage tests

Project roadmap

- **Implementation –** build GS 2.0 components
  - Game server core
  - Back Office
  - Alarms
  - Business Intelligence
  - Demo Casino

Development – Principles

- **KISSS** – Keep It
  - Small
  - Simple
  - Scalable
- Incrementally deliver
  - **long term project**
- Iteration cycle
  - Build, **Prototype**
  - Measure, **Test**
  - Learn, **Tune**
  - **Deliver**

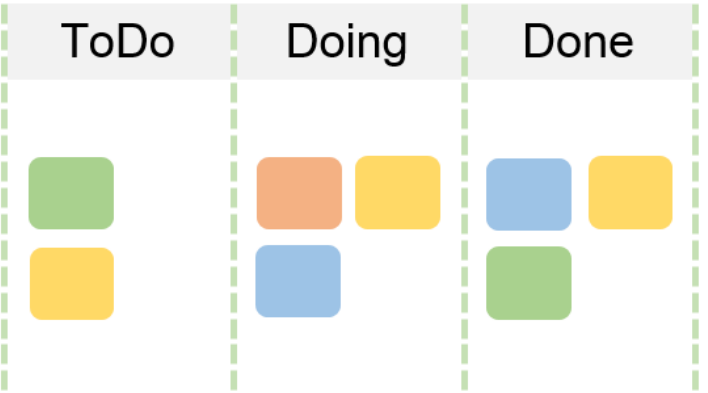


Development – Methodology

- **Scrum** – A fundamental **time shift**
  - Time-boxing doesn't work sometimes
- **Kanban** – A fundamental **incremental improvements**
  - Can be event-driven instead of time-boxed (scrum)
  - Can add new items whenever capacity is available
  - Cross-functional teams, specialist teams allowed

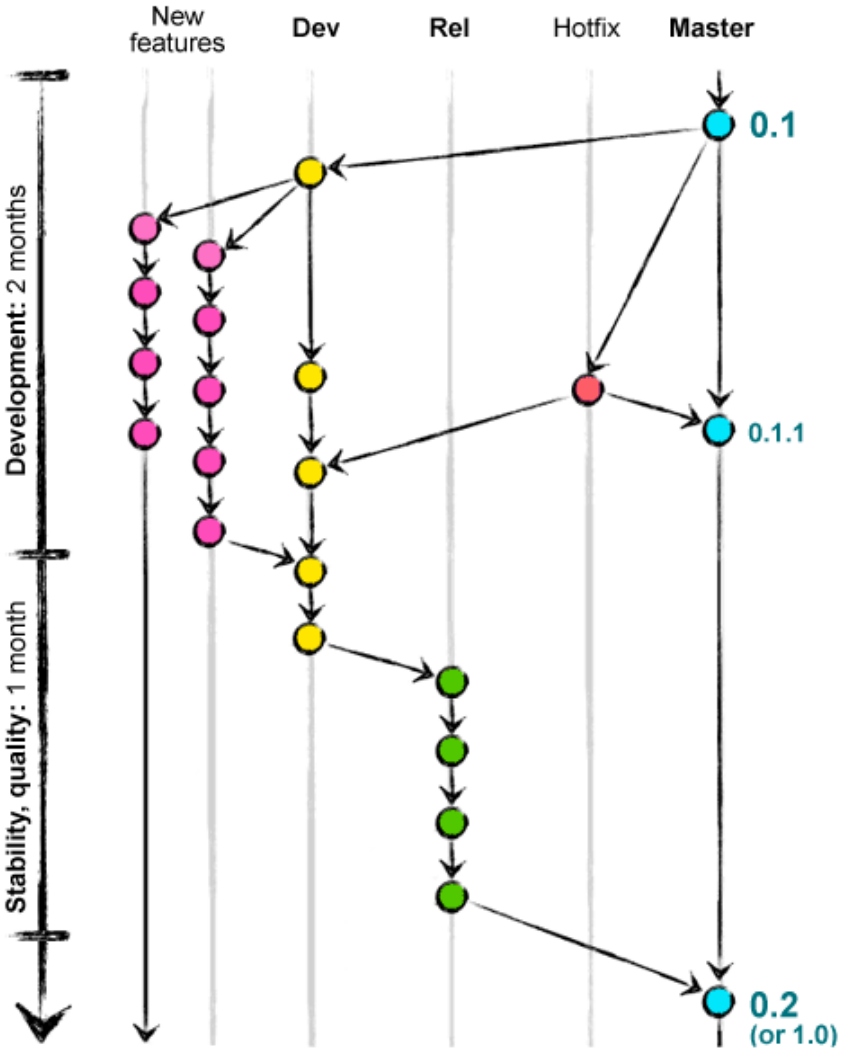
=> Kanban

- suitable for our environment



Development – SCM

- Source Code Management
- **Human gatekeeper**
  - Workflow
  - Gatekeeper does code review
  - Merges the work into the main branch if it meets the necessary standards
  - Strong source control
  - **My best experience**
- Git
  - distributed, offline work



About me – Peter Rybár

- IT Consultant
- Project Leader
- Solutions Architect
- Technology Leader
- Senior Developer
- Certified Scrum Master
- HR consultant

- RNDr., PhD. In Physics
- A-Tech s.r.o. CEO, CTO

http://prest-tech.appspot.com/peter-rybar

- **> 20 years** of IT experiences
- Dozens of IT projects
  - as leader, architect, developer
- Senior solution architect:
  - Softec s.r.o. , Centaur s.r.o.
  - Davinci software s.r.o.
  - VECTARY
- Customers:
  - T-Mobile Slovensko, a.s.
  - Telefónica O2, s.r.o.
  - Orange a.s.
  - Allianz
  - OVB
  - SPP
  - TraviAustria
  - …