

Practical Experience Building a Modern Trading System (Crystal) with Rust & MongoDB

Outline

1. What is Crystal?
2. Architecture/Technology Choices
3. Live Coding Example (Product Management w. Atlas)
4. Future Plans

What is Crystal?

Crystal is a platform for Sales & Trading to allow seamless workflows, scalable pricing/risk calculations and low latency execution.

This presentation will focus on Crystal's workflow capabilities in the Securitization Market.

Brief Background on Securitization Trading

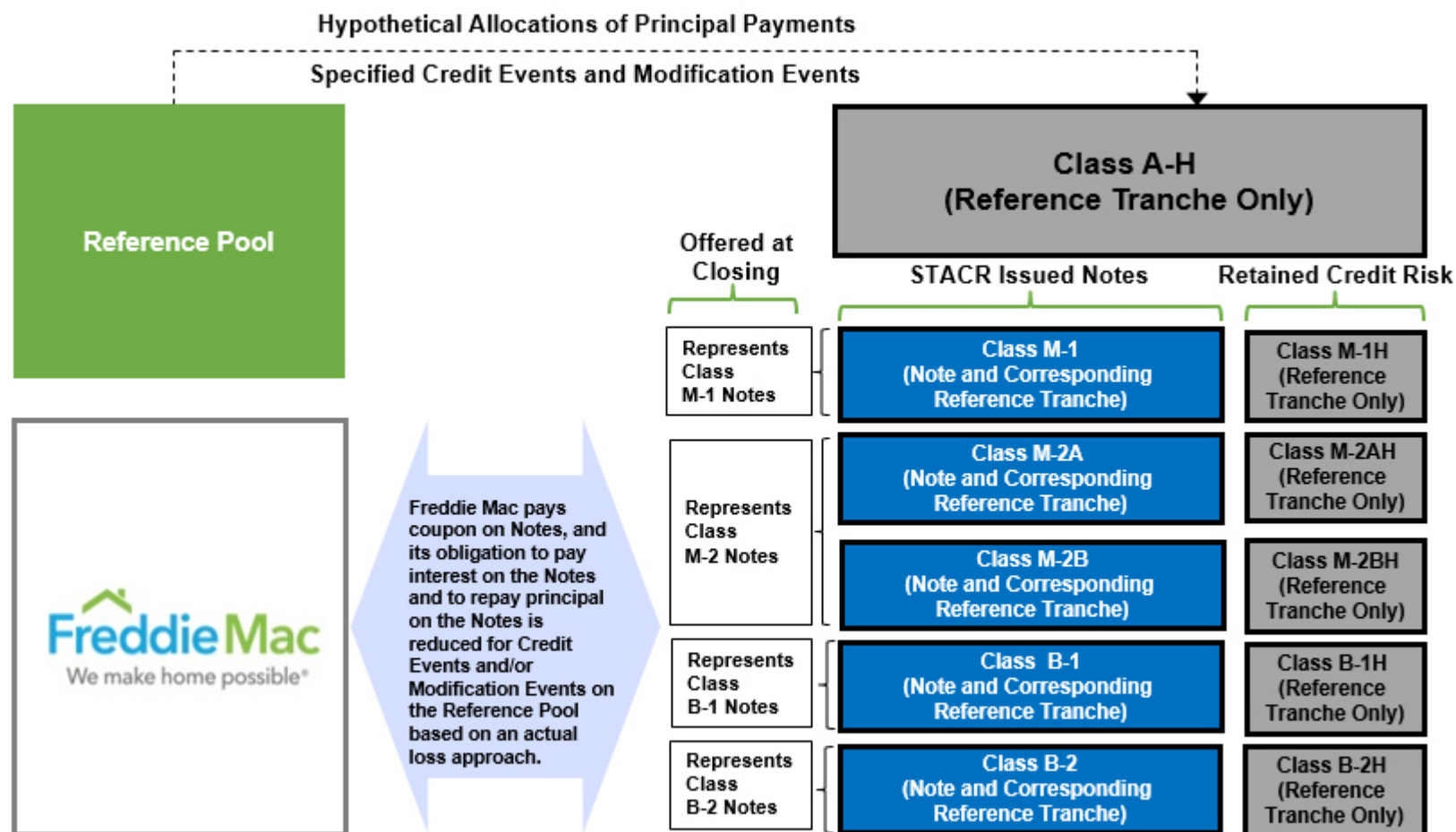
- When you take a series of assets (like mortgages for example) and package them you are securitizing them. For example:

Asset	Size(\$k)
Mortgage 1	52
Mortgage 2	140
Mortgage 3	30
Mortgage 4	225
Mortgage ...	
Mortgage 5,000	126
Total	1,000,000

After Securitization Becomes...

Liability	Size(\$mm)
Security A	500
Security B	350
Security C1	50
Security C2	50
Security R	60
Total	1,010

An actual example:



Freddie Mac may sell a portion of their retained vertical slice, but has agreed to maintain ownership of at least 5% of the M-1 and M-1H, M-2A and M-2AH, M-2B and M-2BH and B-1 and B-1H Reference Tranches and intends to maintain ownership of at least 75% of the B-2 and B-2H Reference Tranches

After these securities are created they trade via:

1. Voice/email Auction: using an informal auction run by the seller called a Bid Wanted in Competition (BWIC). For example:

Id	Security Name	Of	Cf
A9484	Security A	500	140
B8374	Security D	200	30
A9023	Security C	325	325
A1739	Security Z	50	45

2. Private or Semi-Private B2B Trading: Email, Voice or Chat is the main mechanism

Together, these are called Over The Counter (OTC) trading because it is not on an exchange.

BWIC Workflow

Step #	Step	Today	Crystal v1
1	Client Sends BWIC to 1-40 Dealers	Email	Email
2	Trader anonymized and Forwards to Sales	Email	Crystal
3	Sales forwards to their client	Email	Email
4	Trader Analyzes securities (asset history, interest rate scenarios, yield targets, etc...)	Spreadsheets/SQL	Crystal
5	Trader Adds Price Talk	Email	Crystal
6	Sales collected client feedback	Spreadsheet	Crystal
7	Trader Adds Final Bid	Spreadsheet	Crystal
8	Trader sends bids to seller	email	email

Architecture/Technology Choices

- System Requirements
 - Performance
 - "Chat" <75ms response time
 - Analysis of 5mm assets x ~500 columns of data x 30 years of monthly history = ~15TB
 - Real time pivot (<75ms) of 2mm assets x 1 month = ~16GB
 - parallel stream upload of data from our clients
 - Customization
 - Clients specify the metadata for assets
 - Safety
 - Small startup
 - No room for bugs
 - Cost
 - need the flexibility to hire consultants for particular features and release them immediately

- MVP Architecture
 - Meteor (Node/Mongo|Galaxy/Atlas)
 - *from zero lines of code to production in 6 weeks*
- Crystal v1
 - How do we scale to the data requirements listed above?
 - we were confident mongo could scale, but what about the node server?

Non-scientific comparison

Requirement	OCaml	Haskell	TypeScript	C++	Go	Rust
Extreme Speed	n	n	n	y	?	y
Easy to Learn	n	n	y	y	y	y
Extreme Safety	y	y	n	n	n	y

Rust is the first language I've worked with where I am totally relaxed when I write code.

- **Ownership:** The ownership model ensures minimum resource usage
- **Types:** Ultra strict typing ensures the vast majority of errors are caught at compile time

Other benefits:

- **Result/?**: easier to use than Try/Catch -- simple to pass types for specific errors
- **Option**: an absolute dream to work with. better than null or it's equivalents
- **Linking to other libraries**: cargo has all the modern package maintenance features. coming from a c++ background thinks like `csv = { git = "https://github.com/BurntSushi/rust-csv.git", rev = "d0642c500b7ea4b4e19d45aa19f10743063a3f57" }` are amazing.
- **Tests**: Test are trivial to add as they are built into the language.
- **.unwrap()**: it is easy to filter for these and prevent them from getting into production code (forcing the developer to handle all cases unless it is just a test script)
- **functional style**: can easily manipulate data structures w/o loops

```
let product_id_fieldnames: Vec<&String> = product_metadata_map
    .product_id_fieldnames()
    .into_iter()
    .filter(|x| product_headers_map.contains_key(*x))
    .collect();
```

- **serde**: brilliant, fast, type safe parsing library
- **documentation**: docs.rs is spectacular for helping you find the package you need
- **Rust is Rust**: other languages drop into c or asm to achieve speed. nearly all of 🦟 is written in 🦟
- **const**: let is const by default

Things I try to avoid:

- **Arc's:** Not understanding how these worked caused a significant memory leak
- **bleeding edge libraries:** I initially tried (despite it being a prototype) using mongodb-labs/mongo-rust-driver-prototype. It worked perfectly on windows and mac, but when we went to production on linux it was 40x slower!
-

Coding Example (Product Management w. Atlas)

Notes:

- <https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>