



Redesigning Legacy Systems:

Keys to Success / Lessons Learned

Pete Muldoon

Redesigning Legacy Systems

Keys to Success / Lessons Learned

ACCU 2021
March 12, 2020

Pete Muldoon
Senior Software Developer

TechAtBloomberg.com


Questions

#include <slide_numbers>

What is a Legacy System?

Legacy : Current software in production use

System : A set of intercommunicating components which itself may form part of a larger system



Why?



Will a partial rewrite do?

Possible Reasons to rewrite a legacy system

Machine or Software End of life / Switch

- No longer available/maintained
- Costly to maintain
- Better value
- More horsepower

Throughput / latency

- System built when load was much smaller
- New techniques not available or used

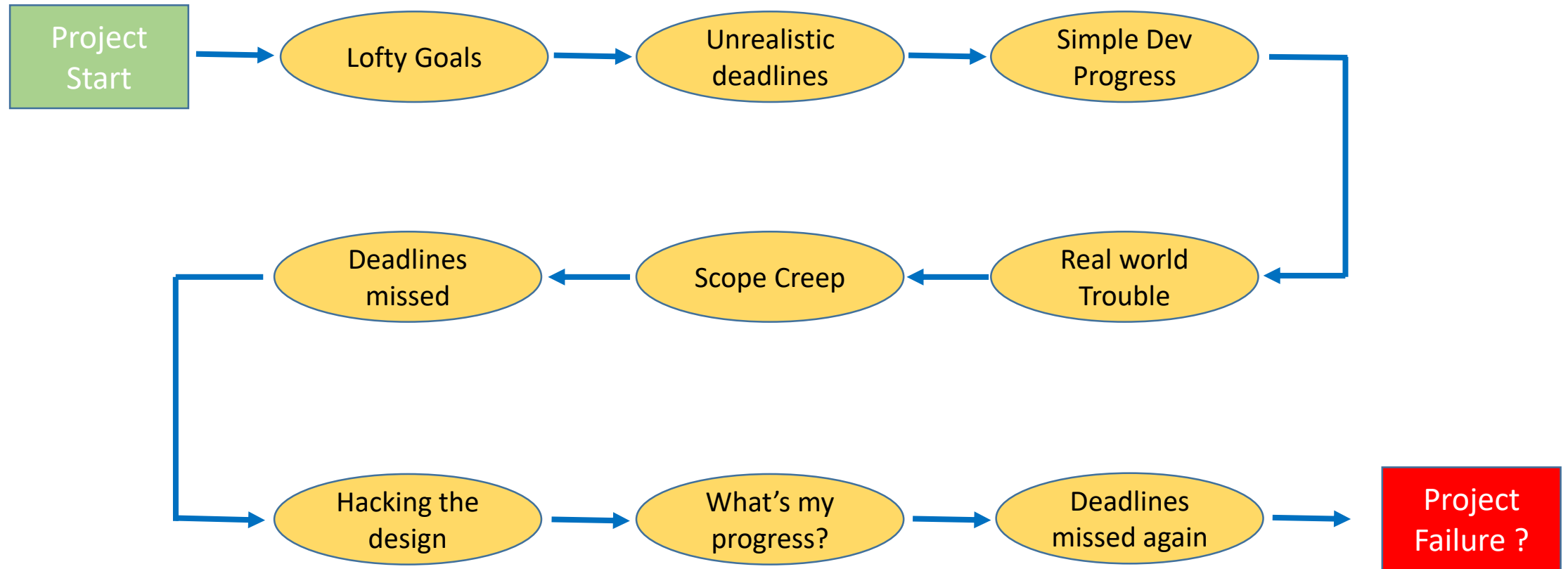
Improve Maintainability / Reliability

- Untangle operational logic
- Remove unused/rarely used operations

Opportunity

- Just because / Executive Order
- One-off events
 - Y2K
 - Linux migration

Word of caution



Key Ingredients: Things You'll need

Definition(s) of Success / Proving Ground

- Is there a problem that will go away when this rolls out ?
- Some metric(s) that will be realized ?
 - Problem is these metrics are usually synthetic and poorly derived

Team Composition

- Historical System Knowledge
 - To disentangle current system
 - Provide context on little-known tricks/hacks that were employed
 - Current Pain points; provide future proofing
- Design Architectural Experience
 - Proven track record
 - Seasoned
- New Blood / Creativity
 - Novel perspectives

Business/Market buy-in

Key to Requirements: Analyze how Data flows through the current system

Gather qualitative and quantitative intelligence

- Peak loads
- Long latencies
- Input composition

Confirm your intuition and uncover misconceptions

- Largest volumes of transactions (probably simplest to replace)
- Apply/Test own measure of value
- Throughput pain points discoverable

Hypothesize outcomes, innovate, and implement design

Map strategic goals

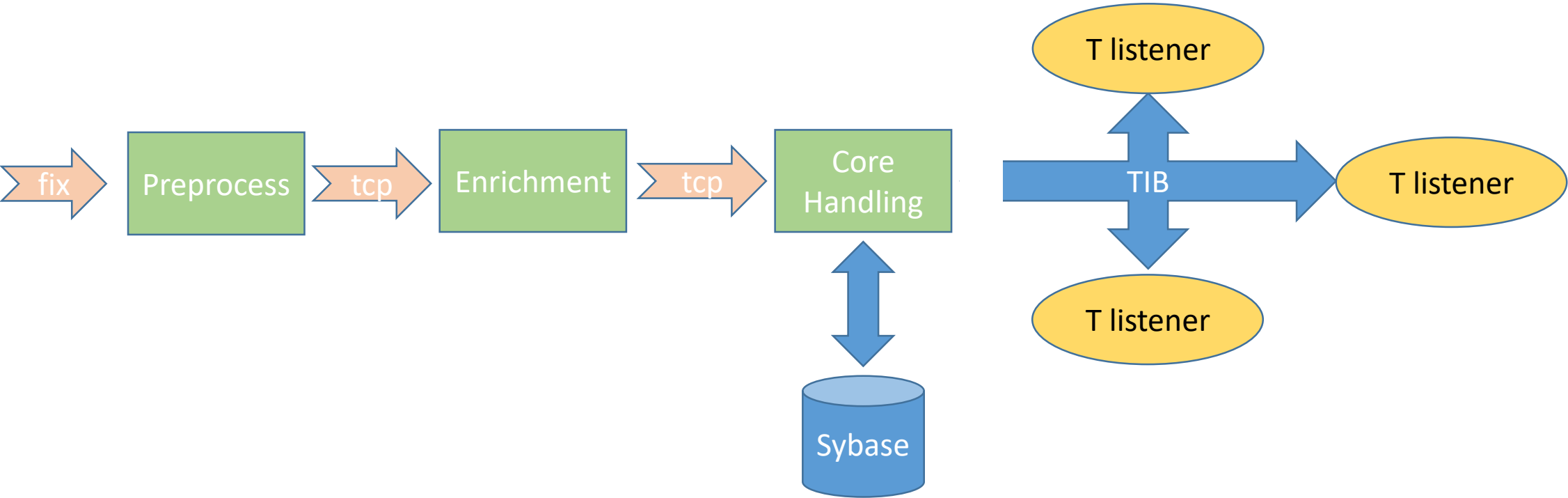
Most under used design asset, can base a design on data evidence

Key to System Design : Design over Technology

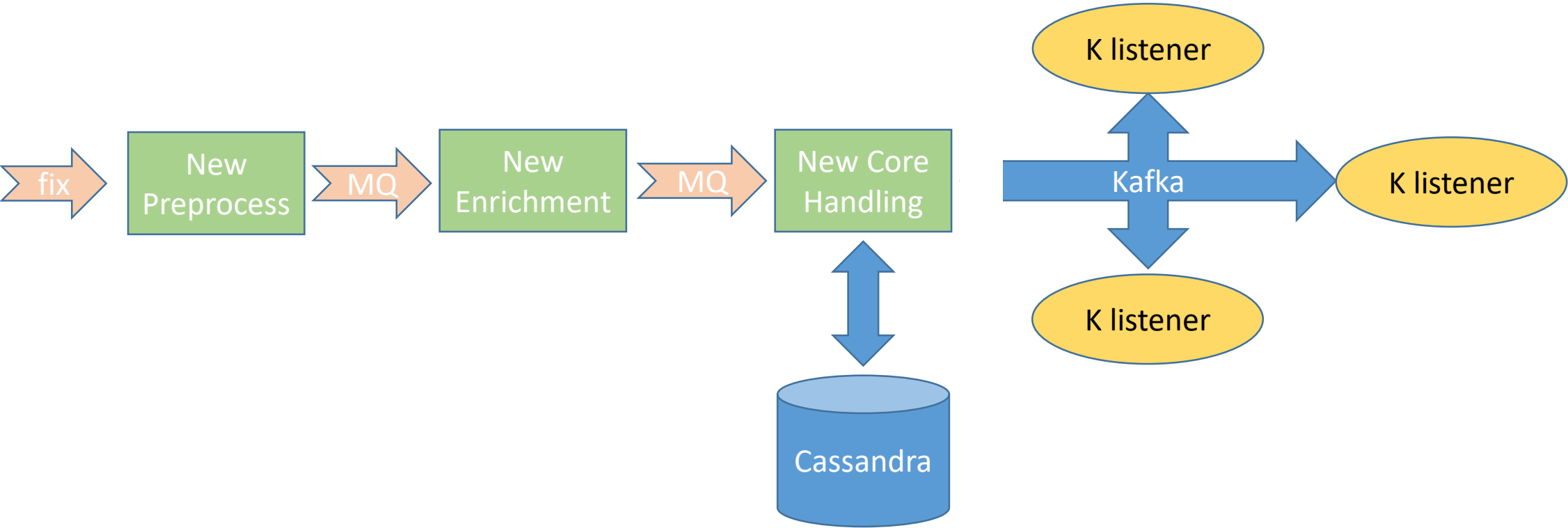
Produce a design first and plug implementation technologies later

- Design is technology agnostic – to a largish degree
- Inclination is to believe technology will solve systemic problems
- Avoid mimicking legacy design with new technologies i.e. a port

Old System Architecture



New System Architecture



Key to System Design : Design over Technology

Produce a design first and plug implementation technologies later

- Design is technology agnostic – to a largish degree
- Inclination is to believe technology will solve systemic problems
- Avoid mimicking Legacy design with new technologies i.e. a port

Underlying technology replacement is usually not the answer.

Using latest technology is nice but leave to end of the design process.

Unfortunately *Buzz words* and *unrealistic expectations* are the norm and wins approval

An Aside about New Technologies

Easy to introduce but hell to live with

- Product Maturity
- Production-level integration?
- New technologies don't advertise problems only benefits
 - Unknown unknowns
 - You'll find the cons when you start using it

Ask “Can I use *current* known implementation technologies in my new design?”

Key to Charting Progress: Project must be staged via Deliverables/Milestones

A **Deliverable** is a measurable and tangible outcome of the project.

Milestones are checkpoints throughout the life of the project. They identify when one or multiple groups of activities have been completed thus implying that a notable point has been reached in the project.

Tasks are small packets of work to achieve the above

But they should be

- Attainable/Realistic – A judgement call
- Observable - hopefully directly via deliverables
 - Additional functionality / load handling
- Chartable
 - Final Prod deliverable is all but useless for charting progress

Functionality/Code Metrics over feelings

Key to Moving Forwards: Iterative/Adaptive Improvement

Quality is the result of consistent incremental improvement

- Realize iteration will approach perfection in a sane manner.
 - Don't borrow trouble with far-fetched scenarios
- Beware Scope creep
- Small tasks (3 - 5 days) that start over flowing time boundaries need to be split into more tasks that can be dependent.
 - Originally misestimated, requirements not understood
- Real world is messy, struggle to adhere to design goals and overall vision
- If current design has a terminal mismatch with reality, evolve the design.
 - Slavishly holding to a paradigm shows shortcomings in paradigm, be flexible

Key to Delivering Product: Shortening the Feedback Loops

Given Iteration is the key to delivering a working system

Identifying and shortening the feedback loop(s) is critical to hitting target dates / prompt delivery

But where are the iterative feedback loops in producing a deliverable ?

- Compilation / linking
- Unit testing
- Code reviews
- Automated system integration testing
- Development Process reviews aka Sprint retro's
- Alarms
- QA
- Dashboards / Metrics
- User/Early adopters
- Production usage – uh oh

Key to Delivering Product: Shortening the Feedback Loops

Given Iteration is the key to delivering a working system

Identifying and shortening the feedback loop(s) is critical to hitting target dates / prompt delivery

But where are the iterative feedback loops in producing a deliverable ?

- Unit testing - Minute
- Automated system integration testing – 15 minutes
- Code reviews - Day
- Alarms - Whenever
- QA - Week
- Production usage - uh oh - Days/Weeks

Key to Understanding System Health : Building in Observability

Four golden metrics

- Latency – Time taken to service a request (response time)
- Traffic/Throughput - How much stress is the system taking, at a given time, from users or transactions processing through the service
 - e.g. How is latency affected by throughput (requests per minute)
- Saturation – overall capacity/utilization of the service (%CPU/RAM/disk net free, queue depths)
- Errors – Rate of failing requests to total requests – assuming requests are well-formed

The above is a good start but not exhaustive; metrics should be tailored to the situation/environment

Potential drawbacks

- Metrics for Metrics sake
- Hard to understand/link the metrics to each other or overall system performance
- In larger systems – overall round-trip measurement is hard as it can span lots of groups normally in isolation.

“Eyes on Glass” should be replaced by *Alarms* on metrics cresting particular levels

Key to Understanding System Health : Building in Observability

Alarm on : “Bad” Errors

- Intentions not met / undesirable outcome not due to the user
- Processes down / multiple restarts
- Dependencies unavailable

Alarm on : Meaningful Metrics cresting particular watermarks

- Retries per minute
- Request latency above X time
- Queue size increases 80% of Max

Applicable to smaller rewrites ?

Key to Successful Rollout/Delivery : Moving Users

Move New users by

- Functionality ?
- Client ?

Where will the tail be and who will be on it ?

- Partition by Function -- will pull large multi-method clients over sooner; see larger benefits realized.
- Partition by Client -- will move low volume/unimportant clients first; large/important clients will not move until near very end

No one is celebrating moving an important client to the redesigned product 5 years after it was first rolled out

Don't backport new system benefits to old system, give people a reason to move

An Aside on Resources

Resource acquisition:

More resources added to speed-up/keep project on track

Each new developer added initially results in a net loss of productivity.

If you throw half a dozen engineers in close to project rollout, the problem gets even worse

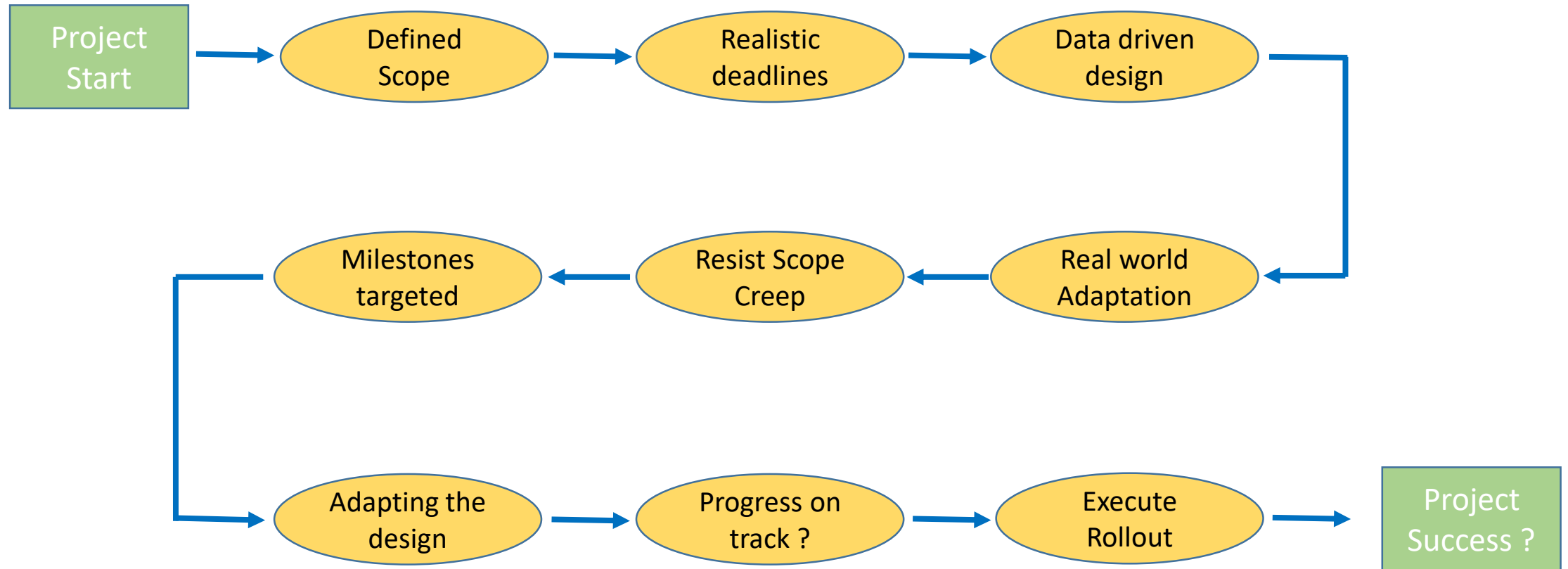
Potential remedies:

- Add developers earlier in the development cycle rather than at the end
- Add Specific Training.

Resource starvation:

- Initial impetus has faded with time or being superseded
- Low visibility

Word of caution



Keys to Redesigning a Legacy System

- Key Ingredients => Have a definite measure of success, Team composition
- Key to Requirements => Analyze the data flows through the current system
 - Biggest volumes of transactions probably simplest to replace, low hanging fruit
- Key to Design => Produce a design first and plug implementation technologies in later
 - Underlying technology replacement is usually not the answer
- Key to Charting Progress => Project must be staged via deliverables / milestones / tasks
 - Attainable/Realistic with Observable effects
 - Keep short task timeframes
- Key to Moving Forwards => Iterative/Adaptive improvements
 - Struggle to adhere to design goals and vision, Remember real world is messy
 - If current design has a terminal mismatch with reality, be flexible, evolve the design
- Key to Delivering Product => Iteration is the key
 - Identifying and shortening the Feedback loop(s) is critical to successful on time delivery
- Key to Understanding/Confidence System Health => Meaningful metrics, watermark alarms
- Key to a Successful Rollout => Move users by functionality not by entity
 - Pull large multi-method clients over sooner; see larger benefits realized
- Key Intangible => ????

PILAR : A Case Study

Brief history :

A system that received, enriched and processed trades to determine whether the trades could proceed to settlement.

Project rewrite initiated when the Legacy system was taking 30 minutes for trades to process through it at market close.

Multiple tickets about the issue from clients on the delay were simply resolved with "Pilar system in development".

No real plan or target completion dates.

Disclaimer : *This example is drawn from my 30+ years of industry experience as a software engineer.*

Key Ingredients : Things You'll need

Definition of success / proving ground

- Is there a problem that will go away when this rolls out ?
Yes, 30 minute processing wait times

Team Composition (4)

- 1 long time legacy system maintainer
- 1 new programmer
- 1 middling programmer
- Myself

Key to Requirements => Analyze the data flows through the current system

Trades from 2 sources : Direct , OMS

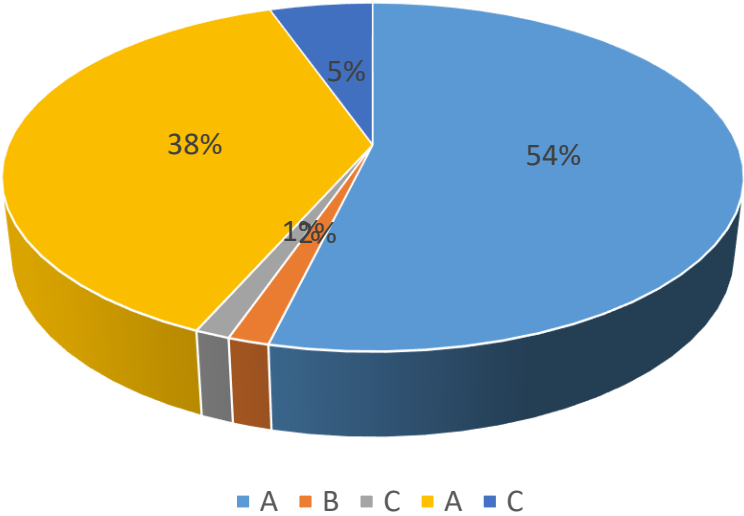
53% direct from clients / 38% from OMS

Trade composition were Types A : 90% , C : 5.25%, Misc. : 4.75%

DIRECT										OMS		Grand Total
Type A	B	C	D	E	F	UNKN	G	H	I	A	C	
275334	7751	6596	5697	1473	646	567	494	19	1	194884	27527	520989

Direct trades

- Top user accounted for 78%
- Top 4 users accounted for 85%



Key to Requirements => Analyze the data flows through the current system

Request latency Analysis

- Lagged in Enrichment Phase
 - All Operations done serially in arbitrary order
 - Any hang-up in any enrichment could freeze system

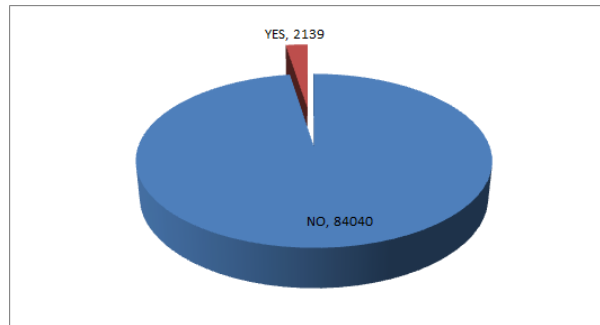
Peak time Analysis

- Throughput lagged due to stampeding volumes/throttling
- Duplicate Trades entered by user – due to lag

Key to Requirements => Analyze the data flows through the current system

Last Snag – Trades where manual intervention is used

MANUAL_USER	count(p_key)
NO	84040
YES	2139



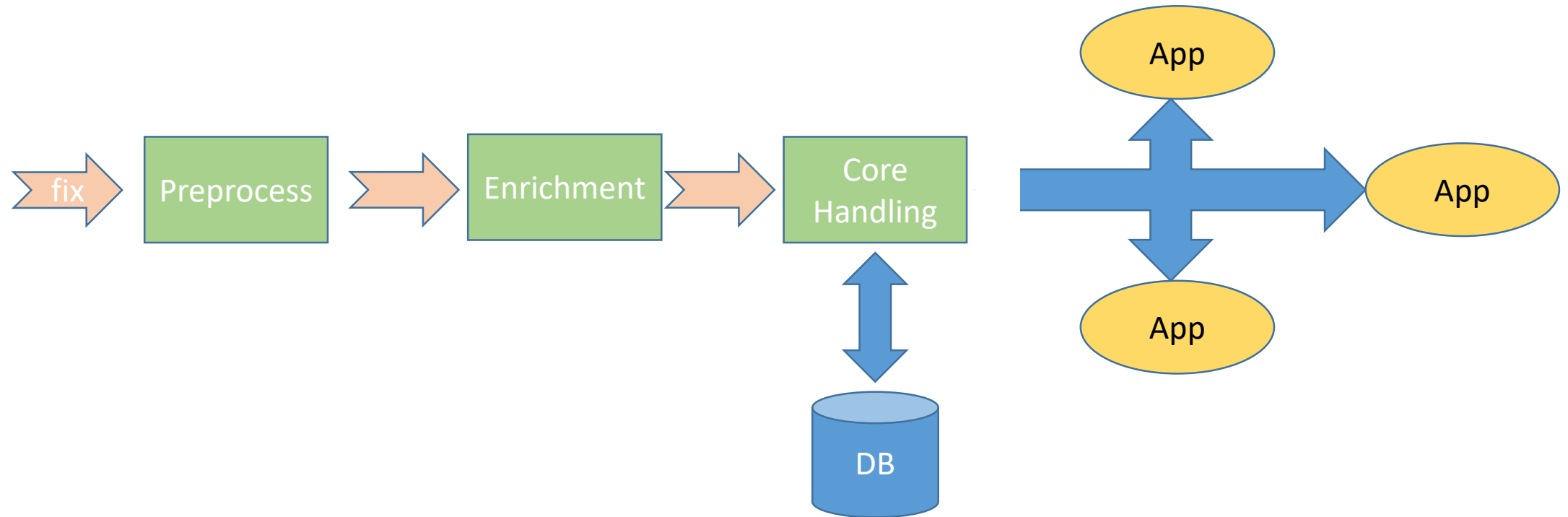
Manual Trades : 2.48%

Simplest trades / largest volume / few clients => Efficient win targeted

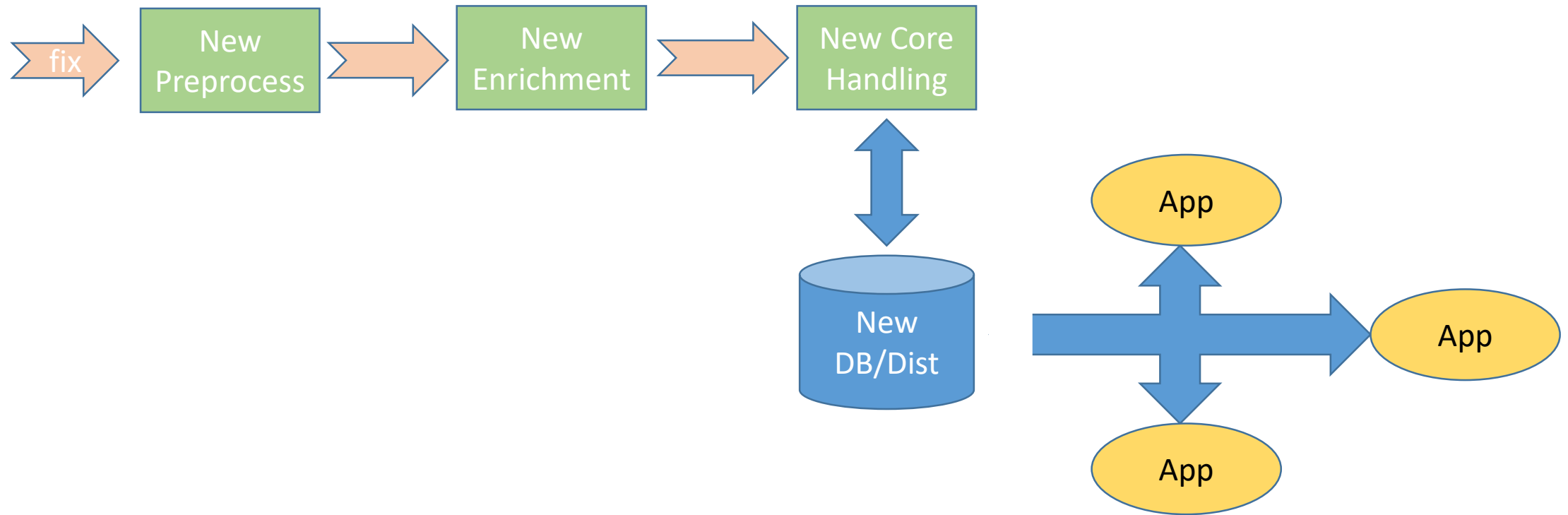
Keys to Design => Produce Design first and plug in technologies in later

Wrong approach initially - “New design” mirroring legacy system : serial in nature and proven problems (enrichments etc.).

Legacy System Architecture



“New” System Architecture



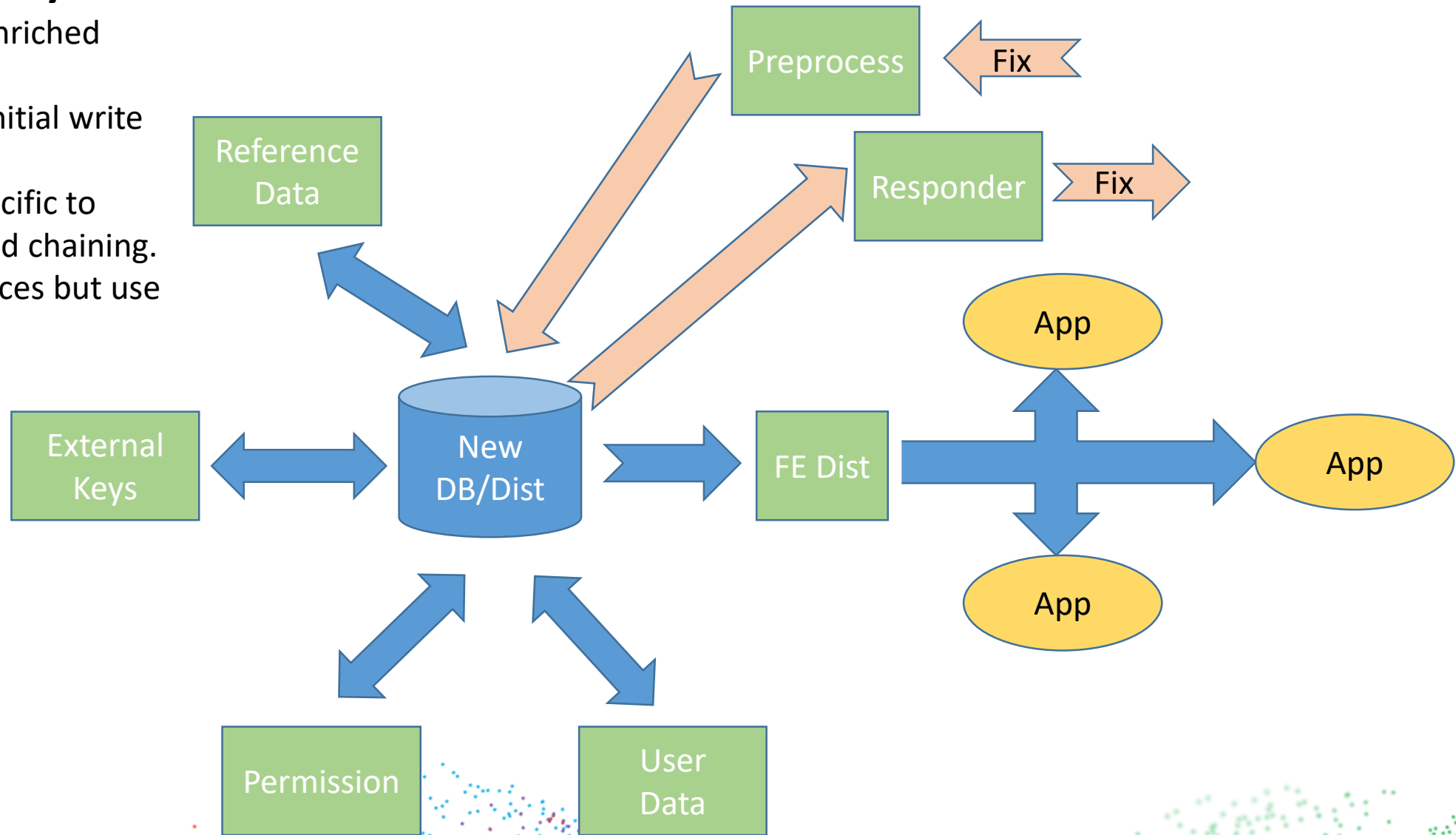
Keys to Design => Produce Design first and plug in technologies in later

Wrong approach - "New design" mirroring legacy system : serial in nature and proven problems (enrichments etc.).

Real new design diagram initiated as hub/spoke model :

“New” System Architecture

- Front-end receives unenriched trade first.
- All enrichment is post initial write & distribution
- Enrichers are highly specific to prevent dependency and chaining.
- All process run N instances but use same configuration



Keys to Design => Produce Design first and plug in technologies in later

Wrong approach - "New design" mirroring legacy system : serial in nature and proven problems (enrichments etc.).

Real new design diagram initiated as hub/spoke model :

- Dependencies lessened/parallel operations promoted
- Scalable
- Technologies an afterthought

Design tackled real world problems based on data analysis

Key to charting Progress => Project must be staged via deliverables/milestones

Milestone 1-x : Replace legacy task copies with brand new code / design starting at task sourcing input data(no enrichment)

Deliverable : Raw unenriched trades showing in app.

Milestone x+1, y : Add N enrichers

Deliverable : Example trades showing progression in app as enrichments i.e. reference data, permissioning, etc.

milestone y+1, z : QA automation of various tests that are applied manually in legacy system.

Deliverable : Regular on-demand QA system testing

Deliverable : Accepted PROD rollout plan

Deliverable : Actual code rollout and clients turned on

Keys to moving forwards => Iterative/Adaptive Improvement

Frequent (cut throat) design reviews esp. on new tasks/concepts/refactors generally cleaned up initial code by approx. 50%

Build for now with extensibility for future. Every conceivable / highly unlikely problems not addressed - cut throat provisioning

Keys to delivering Product => Iteration is key

Design reviews by group

Code reviews by at least 2 people

Prioritize reviews over new work

Unit tests a must for code acceptance

i.e. any enhancement required a unit test

QA system automation allowed high chance for spotting problems and rapid deployment to dev/beta/prod

Keys to System Health => Built in Observability / Alarms

Basic – System Health checking

No Telemetry / Watermarking

Resource acquisition/starvation

Halfway through : Team of 3

Final Phase : Almost 1

Key to successful Rollout => Move users by functionality not by entity

Due to data analysis - large clients targeted for initial rollout after brief smaller client check.

After 8 months of work with a successful code rollout to Production, no clients due to dependency on another team needing work to send trades our way

As luck would have it, wait times had increased further in interim.

Largest client made direct complaint to senior management. Resource fountain opened up aka spigot turned on, trades started flowing.

Congratulatory email; the wait time with new system had dropped from 40 mins to under a minute, clients very happy.

Years later, still running – Does it now need a rewrite ???

Keys to Redesigning a Legacy System

- Key Ingredients => Have a definite measure of success, Team composition
- Key to Requirements => Analyze the data flows through the current system
 - Locate the problems
- Key to Design => Produce a design first and plug implementation technologies in later
 - Underlying technology replacement is not the answer to systemic problems
- Key to Charting Progress => Project must be staged via deliverables / milestones / tasks
 - Attainable/Realistic with Observable effects
 - Keep short task timeframes
- Key to moving forwards => Iterative/Adaptive improvements
 - Struggle to adhere to design goals and vision, beware the real world is messy
 - If current design has a terminal mismatch with reality, be flexible, evolve the design
- Key to delivering Product => Iteration is the key
 - Identifying and shortening the Feedback loop(s) is critical to successful on time delivery
- Key to Understanding/Confidence System Health => Meaningful metrics, alarms
- Key to successful Rollout => Move new users by functionality not by entity
 - Pull large multi-method clients over sooner and see larger benefits realized
- Key Intangible =>



Questions ?

Contact : pmuldoon1@Bloomberg.net