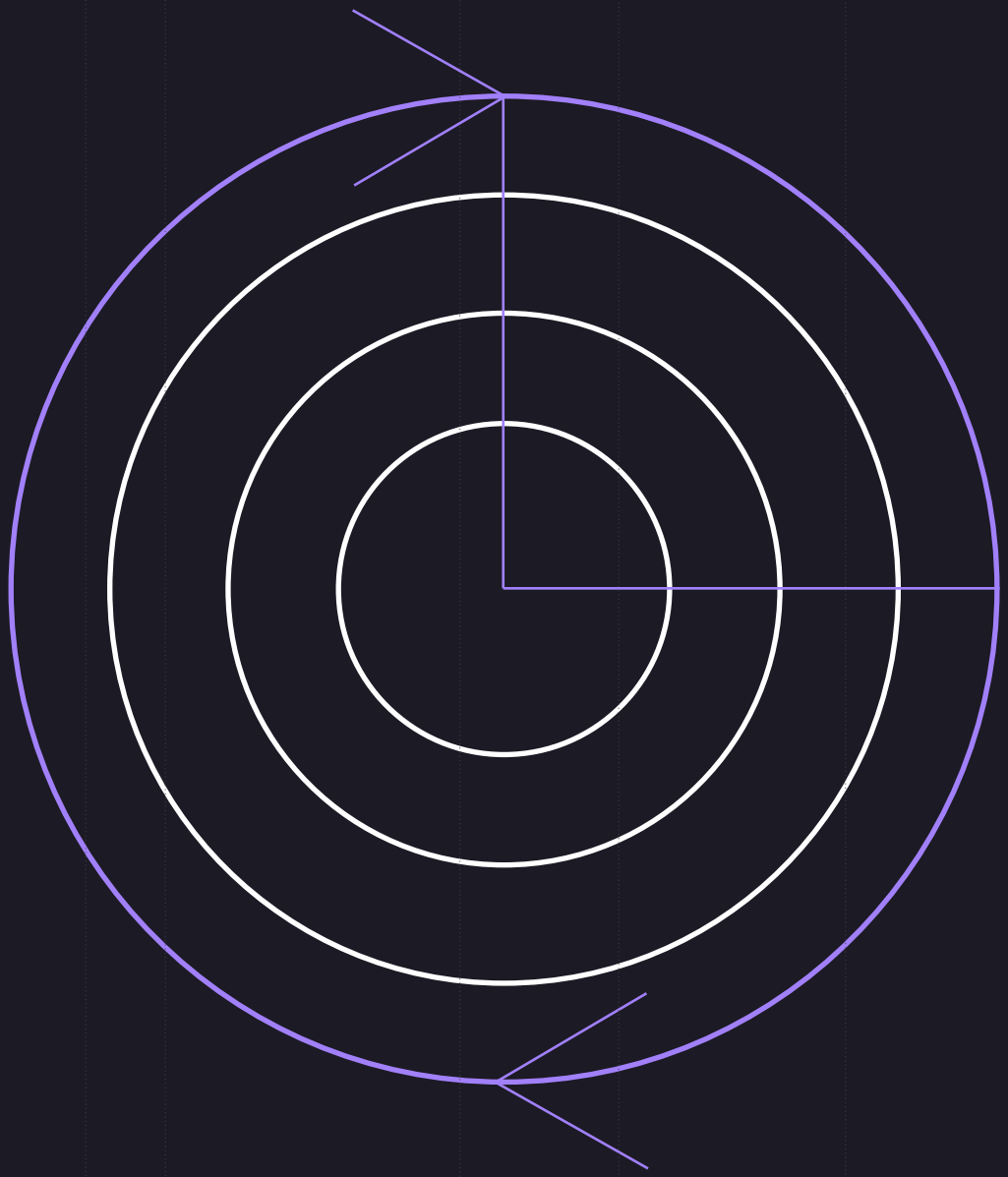
Digital Report 2020

From Batch to Real-Time

The Next Era of Speed in Banking

Financial Services



KIN+CARTA

www.kinandcarta.com

Index

Kin + Carta exists to make the world work better.

Executive Summary

As customer experiences rooted in real-time technology continue to set new standards for speed and convenience, batch release systems present a major obstacle for enterprise financial institutions. Moving from batch to real-time technology can help CTOs, CIOs establish new foundational ways of working that create value across the organization. This whitepaper explores the specific shortcomings of batch processes, benefits of real-time, and specific recommendations on how to implement real-time systems.

Batch to Real Time Key-Takeaways

1. **Customers expect frictionless experiences.**
   Customer experiences rooted in real-time computing are setting new standards for speed and convenience in financial services transactions.

2. **Batch processes are holding enterprises back.**
   Including data inaccuracy, overly complex system maintenance, wasted resources and lost revenue, banking technology that relies on batch processes are slowing the progress of global financial institutions.

3. **Real-time, event-driven architectures create new value across the organization.**
   The benefits of real-time systems include speed-to-value, new product capabilities, consistent communication, more reliable analytics and more efficient testing procedures.

4. **Real-time systems should be pursued within broader Modernization initiatives.**
   Unlocking greater efficiencies and new capabilities through real-time technology can't be done in isolation. Building a fully modernized foundation depends on a holistic perspective of the technology, people and processes that make up the fabric of your business.

INTRO

The Bridge that Banking Needs

Before the Brooklyn Bridge was completed in 1888, the only way to travel from Manhattan to Brooklyn was to cross the Hudson river by ferry. When the plans for the bridge were announced, they were met with great skepticism. Their concern? The bridge will get in the way of the ferries. In addition to doubts around the sheer possibility of the bridge itself, skeptics were concerned about how new structures would interrupt the usual ways of getting around.

It may be nearly 150 years old, but this story highlights precisely the difference between batch and real-time systems; between limited "batch" ferry routes and the unparalleled impact of exchanging goods, services, and people in real-time.

Building new foundational infrastructure requires new mindsets and the willingness to evolve old ways of operating, which is inherently challenging. But it doesn't mean legacy systems are thrown out the window.

For New York City, the Brooklyn Bridge became a catalyst of economic growth unlike any other. For leading financial institutions willing to invest in new foundations of speed, efficiency, and connection with customers, moving to real-time systems could spark the same type of progress.

The Context

Speed & Banking

Since the earliest forms of banking thousands of years ago, speed has always been the underlying driver of progress. Dating back to the first documented examples of money lending, the demand for resources to be accounted for, communicated and exchanged in less time catalyzed new innovative processes and products. Today is no different. Accomplishing virtually any financial task from your phone has been cemented as the normal. But as global tech juggernauts, established financial institutions and smaller tech disruptors all compete to deliver faster, more reliable customer experiences, entrenched batch processes are proving to be the most significant hindrance to the next era of speed in financial services.

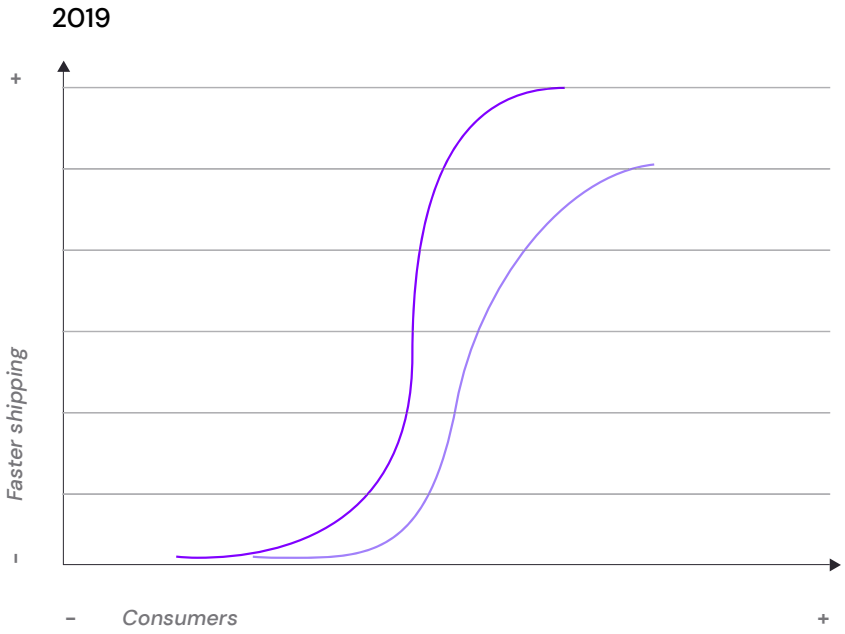
"Building new foundational infrastructure requires new mindsets and the willingness to evolve old ways of operating, which is inherently challenging."

Beyond Banking - The New Normal

It's hard to imagine a world where shipping fees and 7-10 day delivery windows were the benchmark of great online shopping. Since Amazon Prime first launched in 2005, it created a new standard of possibility in delivering consumer goods. In April of 2019, after announcing a new one-day shipping guarantee for Prime members, Target and Walmart shares both took a tumble, leading to Walmart's "NextDay" shipping guarantee just a month later. Recent research conducted by Salesforce even found that 58% of consumers stated they would not buy from a company that can't ship quickly and cheaply.

The real estate industry is a great example of new expectations of speed being solidified by digitally-native companies like Redfin and Zillow. In May of 2019, Redfin introduced a new service allowing home sellers to solicit offers directly through the web, chipping away at the friction caused by unchanged legacy processes.

From a technical standpoint, APIs have risen to the level of being products of their own. Not having real-time APIs used to just prevent companies from enabling employees to build modern and innovative experiences. Now a lack of real-time APIs represents lost profit and fewer partnership opportunities. The more your data is real-time accessible, the more valuable it is.



"58% of consumers stated they would not buy from a company that can't ship quickly and cheaply"

Evolving expectations in financial services

New milestones in frictionless experience continue to shape what consumers expect in everyday banking. While there are already a handful of credit cards that offer the ability to instantly be able to spend on the card online after you are approved, the Apple Card is the only card designed thus far to spend instantly at a physical point-of-sale. Apple moved into the credit arena with the Apple Card that allows you to begin spending in-person and online in the same minute that you are approved via Apple Pay.

Venmo, Cash, and latecomer Zelle have set an expectation that your money should be able to move around faster than ACH transfers. Zelle gets around slow bank-to-bank settling of funds by creating an agreement between participating banks to make transactions instantly available, even though the money hasn't actually been settled on the backend in the banks' batch-processed ledgers. Customers are flocking to these technologies, too: 75% of millennials report using instantaneous P2P payments and P2P payments are used in many aspects of customers' lives.

Paying rent, paying for meals, splitting utility bills, and giving gifts were the top use-categories for instantaneous P2P payments last year.

From depositing a check using your phone's camera to opening up a credit card in your Apple Wallet, expectations in financial services have become more self-service and digital in the last 20 years. The ability to transition to real-time operating models will be the key in responding to market pressures for speed, reliability and disruptive new customer-driven products and services.

The goal is not to go real-time for real-time's sake. There needs to be a business driver to make that change worthwhile. As Google and other technology firms move into financial services products like checking accounts, expect them to target any area of the consumer experience that includes delays like these.

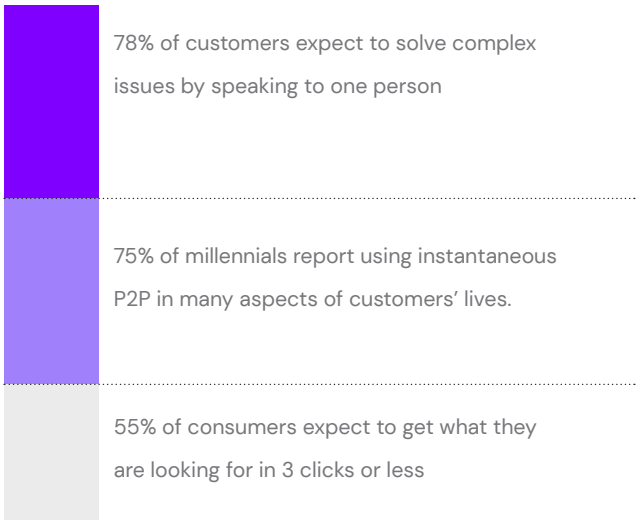
The Business Case for Real-Time Systems

This all sounds great, but all these technical advancements are meaningless unless they drive real business outcomes. By moving to more real-time event-driven systems, customers can see tangible benefits.

Friction-free customer experiences

Foremost, end users should see major processes happening sooner. Things like getting instant approval, up-to-date account information, and quick feedback on service requests are important in today's environment. In fact, 55% of consumers expect to get what they are looking for in 3 clicks or less. This extends notably to customer service as well. Calling a customer service with a change that was just made, or an error that occurred, is frustrating enough.

A Glimpse into Banking Expectations in 2020



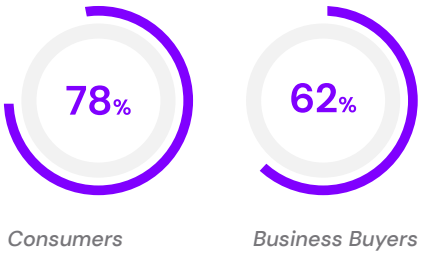
Adding inconsistent system states and the need to call back later after your request or error mitigation processes run through a batch process is not only frustrating to the end user, but it falls short of current customer expectations. Ultimately, that is costly to your company. In a customer-centric world, where 78% of customers expect to solve complex issues by speaking to one person, a real-time system that shows a coalesced state of the system, and raises up errors and other triggers immediately is important in handling issues promptly and professionally. Beyond falling short of expectations, additional calls to customer service cost companies valuable time and money; the call center industry alone was valued at 210 billion dollars last year.

Speed to Value

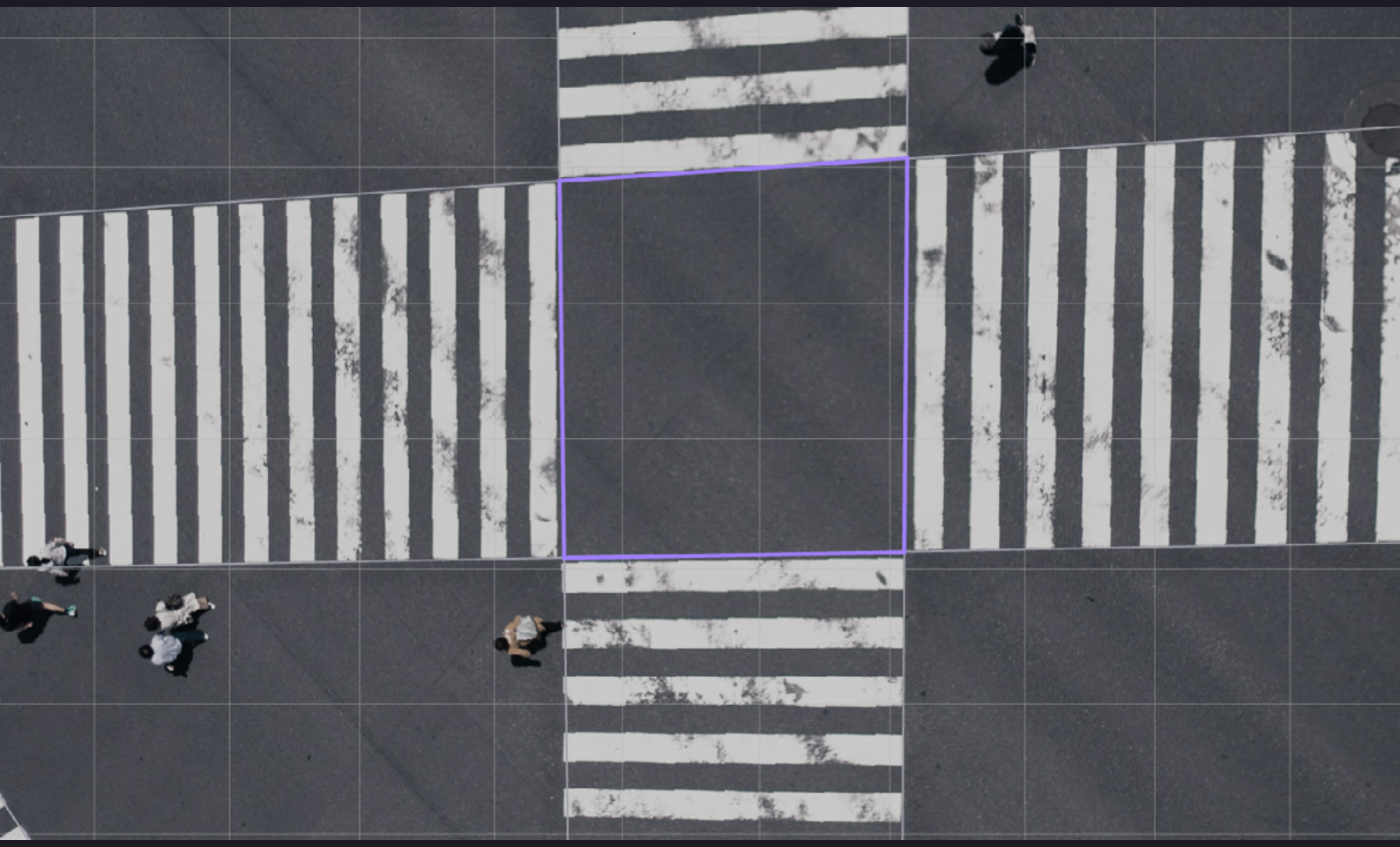
When institutions move from batch to real-time systems, their end users also see benefits in increased rate of innovation. In a world where 73% of millenials, 68% of Gen Xers, and 57% of Baby Boomers expect companies to deliver better products more frequently than in the past, this is becoming increasingly important. To that point, event stream processing is growing rapidly; the need for more precise, informed innovation is pushing companies to use data from their event streams to direct their innovation and better please customers. Integrating into a legacy batch-driven system typically involved weeks or months of requirements gathering and prioritization, painful integrations necessitating getting scheduled into the proper testing environment, before finally waiting weeks or event months to deploy your change in a change window. Because event systems decouple the event producer from the consumer, consumers can integrate functionality much more quickly.

Keeping Pace

A customer's perception of an organization can depend on how they use technology, too. Today, 62% of consumers and 78% of business buyers say that the way a business uses technology reflects how they perceive it to operate overall.



"The way a business uses technology reflects how they perceive it to operate overall"



Problems with Batch Procesing

How did we get here?

The concept of batch processing arose out of the business systems of the 50s and 60s when computing power was low, many systems were still based on 80 or 96 column physical cards, and it wasn't possible for computers to do more than one thing at a time. The computer was instructed via cards - control cards, file input cards, program execution cards, in fact an entire job control language that still exists today was organized around telling the computer how to do a "job". Those "jobs" were batch jobs - everything was a batch job. Transactions? A stack of cards created on a keypunch machine.

Master files? Another stack. A ledger? A stack. On and on. When a stack was dropped on the floor, there was a panic because there might not be time to put it back in order before the next job needed to run. Nirvana? Hardly, but businesses ran on this model for many years.

Eventually cards moved to other media, but the core concepts behind the batch of cards, well, stuck. Any sequentially ordered file is nothing more than a virtual batch of cards.

As computers evolved, they gained the ability to do more than just one job at a time. Green screens and "online" programs were created - more user friendly apps that allowed users to view and enter transactional data - only to have those transactions held and processed later - by (you guessed it) a "batch". This was a delightful and amazing user experience...in 1976. In today's world, 75% of customers expect companies to use technology to create better experiences including more immediate, responsive ones.

Over time, the landscape evolved to include databases and many other advancements to make systems more "real time", yet the batch concept persists. The reasons are many - daily maintenance, backups, reorgs; customers not transacting business off hours; lack of any real imperative to get rid of the "batch" concept. The real world has what appear to be batch use cases as well - (name them here from the research - i.e. ACH, account postings, trades settled on major exchanges are still either T+1 or T+2 in many cases). Interesting Q: Do modern fintechs employ batch anywhere? Any typical use cases?

Falling Short of Expectations

We get it - batch systems still work, even today. While batch might still be a "thing", your customers always expect something better - a real time experience. Imagine buying something on Amazon today only to wake up tomorrow to find an out-of-stock notification in your email. Kind of like going to bed with $100 in your bank account only to get an overdraft notice the next morning because some debit posted overnight that you weren't aware of. This is the batch customer experience and no one likes it - IT teams included.

Historically, batch processing ran in isolation - it was the only thing running, hence there was little concern for causing an impact on any other parts of the system. This strategy is still used today - sometimes the system has to be taken "offline" or made unavailable to users, in order to avoid resource contention. In other instances, there have been workarounds created so that there is still some (albeit degraded) level of service available to the customers while the batch runs. For example, at one major clearinghouse, when the end of day batch starts, a separate shadow system is spun up to allow some limited after hours trading to continue. All of the work in the shadow system is in effect a memo posting - it won't be merged into the main transaction ledger until after the batch is completed and the system is prepared for the next day of trading.

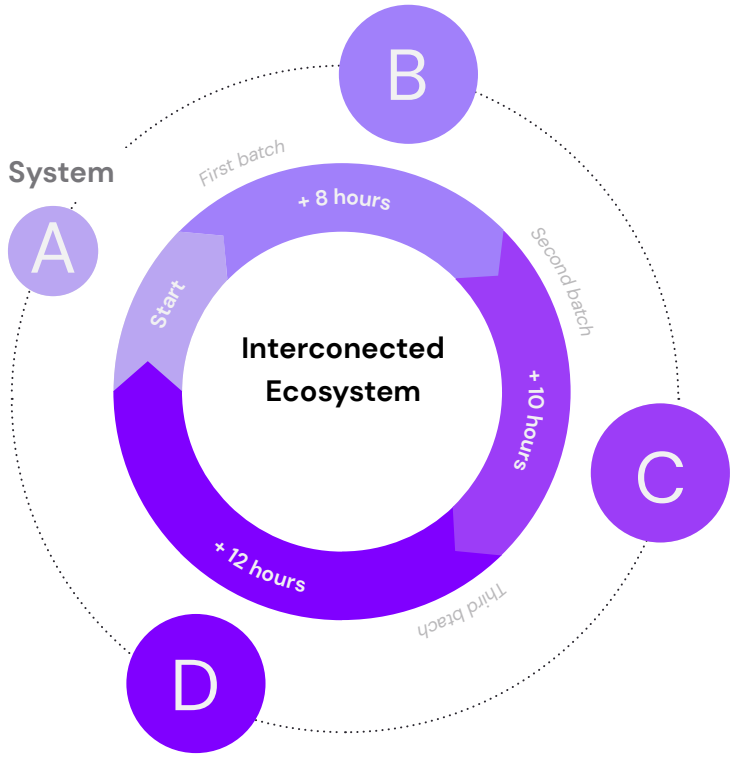
"Imagine buying something on Amazon today only to wake up tomorrow to find an out-of-stock notification in your email."

The Batch Domino Effect

The interconnected nature of today's systems also means that a nightly batch running in one system may also affect other systems - proliferating batch-like behavior all over the ecosystem. How many systems need to be taken offline? How dependent is System B on the outputs of the batch from System A? What about Systems C, D and E? The dance among your batch ecosystem is probably highly orchestrated and managed by only a few people who know all the intricacies, and who are relied upon at 2am to help solve any issues that might arise. This brings us to another issue with batch - the length of time between when a problem is introduced into the system, to the time it is actually detected. Imagine this scenario - a transaction of some kind is entered in the morning, and appears to be correct, but isn't processed until the evening. 8, 10, 12 hours later, the batch is halted and flags that transaction as the culprit. The person responsible for the transaction is long gone, snuggled into their bed for the evening. Now we need to figure out what to do in order to resolve the error.

There may be a standard protocol, there may be reports generated so that the transaction can be repaired on the next business day, or, worst case, everyone is going to get woken up and on the phone so it can be dealt with and allow the batch to finish.

Interconnectedness adds a layer of complexity when it comes to testing too. Oftentimes, batch systems weren't designed (or if they were, entropy has set in) around the concepts of domains and separation of concerns. Contracts between systems usually don't exist, or at best, only informally. Strict rules of encapsulation get broken over time: System A is reading System B's database directly; and System B reads System C's database. Any change made to System A necessitates testing System B, similarly, any change to System B requires a test of System C. Sooner or later, any change - anywhere, turns into a full regression test of every system in the ecosystem because no one is sure whether any change will cause an unintended side effect.



"How many systems need to be taken offline? How dependent is System B on the outputs of the Batch from System A?"

Innefficient Resources

The last complication we will discuss has to do with the computing resources needed to run the batch. Typically these are statically allocated and sized to accommodate the highest expected volume of transactions possible. You might use the historical high watermark and multiply it by two for example. And to size your servers, think classic vertical scaling - when the server runs out of resources, you simply get a bigger server. In reality, you probably get two of them, one for your primary site and one for your DR site. If you have a virtualized environment, maybe that same hardware can be used for other things when the batches aren't running, but if not, maybe it sits idle. In either case, your infrastructure purchases are no longer commodity - you have a bespoke class of infrastructure dedicated to running your batch jobs.

We could spend more time discussing the issues with batch. There may also be positives, but those are probably technical or even romantic. The bottom line is, your customers don't want a batch experience and your employees don't want one either. (Insert Indeed Posting for a Java Batch Programmer. Show that its 30 / 300 days old and has two responses.)

Event Driven Architecture: Putting Real-Time to Work

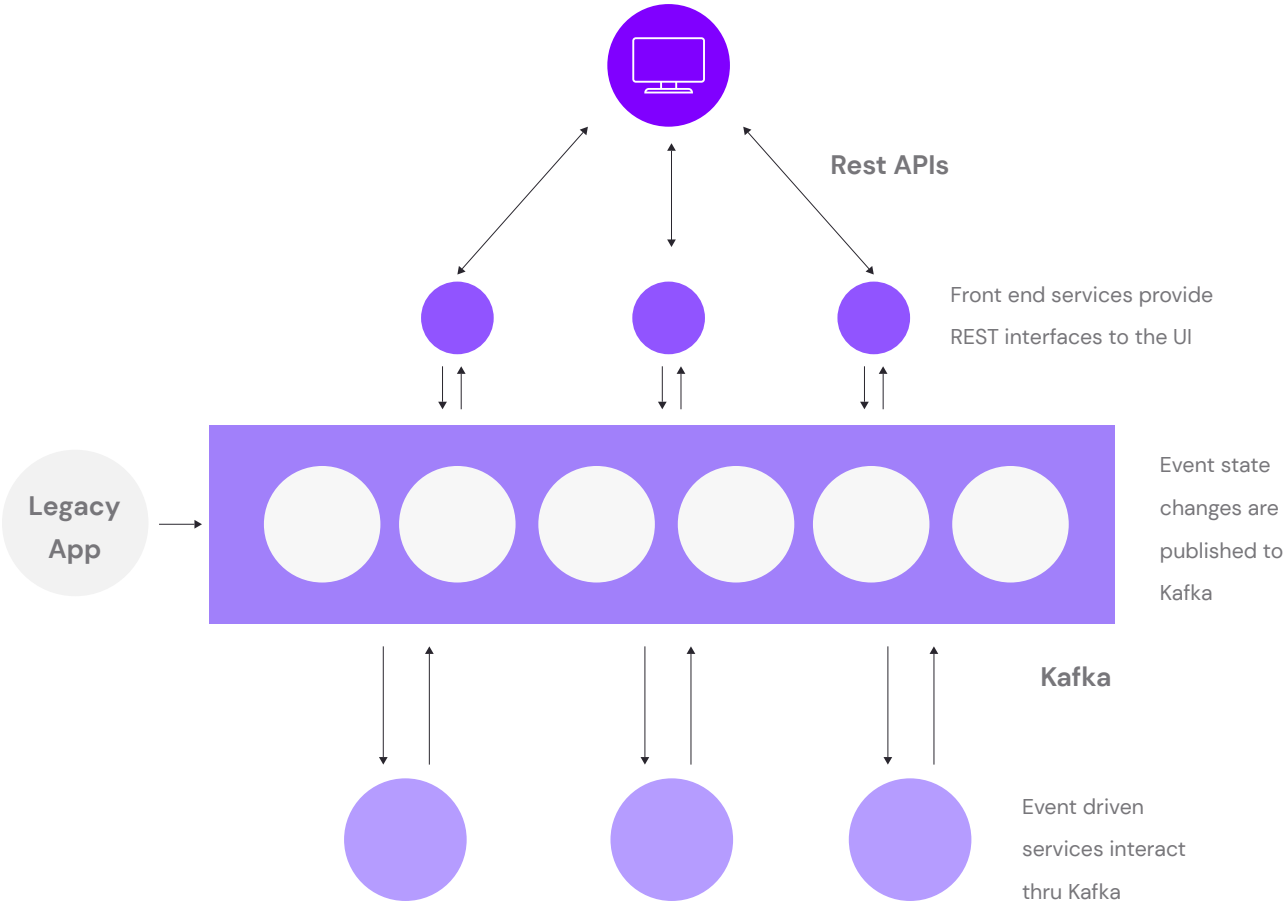
What if we could process information and transactions closer to real time instead of batch? What would that kind of system look like? Modern systems are embracing the concept of an Event Driven Architecture - where "events" are any changes of state whether internally or externally generated - and those events are communicated to all interested parties, as they happen. The interested parties here are services, which represent the basic processing units of the system. The service "listens" for any events it finds of interest, and processes that event. In the course of processing, side effects are generated, representing internal changes of state to the service - and when it finishes processing that event, it publishes an event of its own, letting every other service know what it just did. Similarly other services listen for those events, do their own processing, and send their own events. This process of event consumption, state change, and event publication continues over and over again as a cascade of events moves through the system from one end to the other.

Here is a simplified example based on a trading platform: let's assume we have an Account service which listens for any "trading order" events. The service gets an "order requested" event. It should then check its internal state (whether that's in memory, some file, or a database). It might check to see if this is a duplicate of a pending order already existing in the account, whether the account can cover the expense for the order, or any other checks it might need to make. Assuming the account service has validated the order request, it will save a local pending order in its Account data store, and then it will publish

an “order validated by account” event for other systems to consume. Or, if validation fails, the Account service would publish an “order validation failed” event instead. Other systems (such as the one that originated the order request) will listen for these events and react accordingly.

Here’s an example: an Inventory service listens for all “Order” events. Let’s say it receives one for 17 green widgets. The service checks its internal state (whether that’s in memory, some file, or even a DB of some kind, is beyond the scope of this example), and should it find sufficient inventory, the service will reduce its inventory by 17, and publish an “Order Filled” event letting all the other services know that the order for 17 widgets can be filled and the next steps (i.e. fulfillment) can proceed. If the inventory is not sufficient, then an “Order Backlogged” event would be published, to let everyone (including the originator of the Order) know the item was not available.

An example of event driven services. Legacy systems export data as events to the Kafka message bus. User facing services provide REST APIs to the UI. State changes are published into Kafka as events. At the bottom, services collaborate to perform business processing.



Gaining Efficiencies

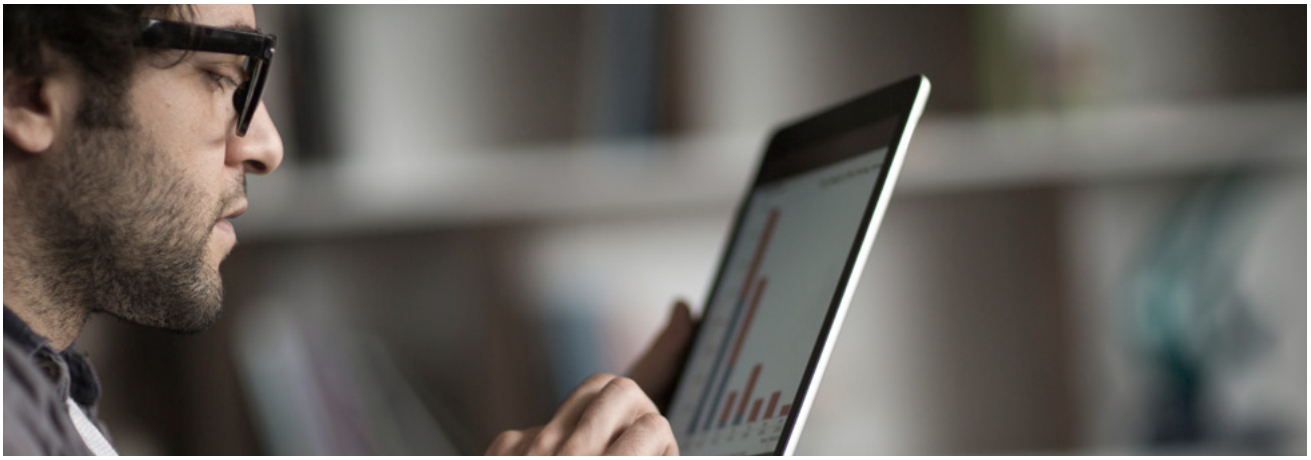
The benefits to this type of structure become pretty evident. This architecture allows for acknowledgement and processing of business events in seconds, instead of hours and days. And since every large enterprise is composed of n-layers of business systems, the benefits of real-time trickle through to provide huge advantages. This speed to processing also translates into speed to remediate issues. Instead of waiting overnight to see thousands (or millions) of events fail due to a processing error, these types of errors can be caught earlier, for a smaller subset, and addressed before they become widespread.

Consistency of Data

Finally, this speed to processing translates to better consistency of data across the enterprise, a characteristic becoming increasingly important as 78% of customers expect consistent interactions and information across all departments. In the example above, many large enterprises might have a front-end web platform, along with one or more separate account and trade fulfillment systems, and finally one or more analytics databases with consolidated data. In a batch-based paradigm, these systems may only become consistent on an hourly or even nightly basis. In an event-driven paradigm, these systems all become consistent within minutes.

Implementing Event Driven Architecture

The simple example above is not meant to imply that a distributed, event based architecture can be implemented without careful thought and planning. On the contrary, attention needs to be paid to many of the non functional aspects of the system that may have been taken for granted in the former batch world - but need to be handled differently in an event based system. For one, every service is an autonomous unit, usually working asynchronously from all other services. As long as the service can consume and produce events, it has no direct dependency on the other services of the system. This means there are less forms of coupling in an event based system, because the architecture naturally enforces the looser coupling of messages (which represent the contracts of a service) vs. each service relying on the internal data or logic of other services. Secondly, this autonomy means that if one service fails, others can keep processing. Sure, there may not be any events to process from the service that is down, but we can focus recovery efforts on only the service that needs to be recovered. And since all events in the system are persisted, as soon as the failed service is recovered, it can simply pick up from where it left off. Compare this to the legacy batch world. If anyone remembers the complexities of 2-phase commits, transactional processing engines, or SOAP transactions built on top of services buses, you know how fragile these systems are. Those synchronous architectures in theory provided for strong data consistency in distributed systems; in practice, they were fragile, complex to implement correctly, and not scalable for current workloads. Event-driven systems, although requiring their own care and attention, are far more resilient.



Eliminating the Batch Blindfold

Speaking of those events, when taken as a whole, they represent an immutable log of everything that has happened in the system. Every input, every output, and every change of state in any of the services is recorded in the log. What can this log do for you? A lot. The log can be replicated to other geographic regions facilitating multi instance or redundant processing - facilitating High Availability and making the classic Hot/Cold approaches to DR obsolete and enabling either Hot/Hot or Hot/Warm with near instant failover. (Bye bye DR). Gone are the days when you have to wonder "what the heck happened?" No more bolting on an audit log as an add on, or turning on "debug mode". No picking through database logs. It's already there. The events in the log can be replayed when needed - great for debugging purposes, or testing, or analytics. Or to facilitate downstream reporting by other services. Or for many other things. The event log is the blockchain of your system: immutable, permanent, and pervasive. This event log also has one other advantage: it can be easily integrated by consumers. Where synchronous REST API's would have to carefully tune themselves against the number of API requestors, event-processing separates the producer of the data from consumers. Producers of events simply need scale to the events they plan to consume, and consumer connections to an event log system can typically scale well.

So how do we ensure that such a vital part of the system like the event log isn't a single point of failure? By backing it with an industrial strength technology like Kafka, and using facilities like replication to ensure there are always more than one copy of the log records securely written to multiple (3+) active replicas. Lose a server? Fine, there is another one in the same data center. Lose a data center? Fine - there is a copy in another data center. Lose every data center in the region? No problem, we have copies in other regions. These facilities are built-in and available to be exploited from Day One with the cloud native infrastructure that today's modern, event based architectures are built upon.

Maturity Model for Increasingly Advanced Outcomes From the Use of EDA

Tactical (Isolated Activity)		**Strategic** (Federated Activity)		
				PERVASIVE **Organization captured a critical mass of ecosystem business events for business-critical continuous intelligence, innovation and scale.** EDA design skills are combined with the request-based design skills and are widely available across the organization. Event streams are monetized through business capabilities marketplaces. Event governance, analysis and decision control are essential parts of the organization's business activity. Participation in business ecosystems and platform business model is advanced by organization's EDA competence.
			ADVANCED **Integrated with stream analytics, AI and API marketplaces, EDA helps enable new leading-edge solutions.** Central event authority provides EDA training and consulting, supports an event marketplace, manages a central event store, and collaborates with the analytics and AI teams. Real-time decision support and automation flourishut-ilizing increasing number of available event streams. Ecosystems are formed or extended by mixing internal and external event and data sources and consumers.	
		CENTRALIZED **Strategic use and governance of EDA is promoted by cross-organization IT leadership.** A central event authority team is formed to provide support for development and governance, promote EDA design, and work with the business to define event semantics. API and EDA designs are coordinated. Once available, event notifications are spreading, picked up by new suscriber applications.		
	BROKERED **Shared dedicated event broker technology is adopted in parts of the organization.** Some IT teams recognize the differentiated capabilities of EDA and begin promoting the principles of event-driven design across initiatives. Lack of experience or tools for development or governance complicated and slows adoption.			
INCIDENTAL **Some isolated use of EDA mostly supplementary to other activities** Some notifications from applications, databases (CDC), IoT and the web are captured and processed. No general event processing strategy.				

AI = *Artificial Intelligence,* **CDC** = *Change Data Capture,* **EDA** = *Event Driven Architecture,* **IoT** = *Internet of Things*

Source Gartner ID: 398057

Organizations find themselves at different level s of preparedness and maturity with event-driven design. Neraly all are at the "incidential" or "brokered" stage: that is, on the receiving end of some event notification activity emanating from applications or databases (incidental); or using a pub/sub capability of message-oriented middleware (MOM) or an enterprise service bus (ESB) for some isolated event-centric application (brokered).

Typical Real-Time Use Cases

Event-driven systems can be helpful for replacing a wide variety of batch-based systems, but are particularly useful to solve the following use cases:

Communication Across Business Domains

Event-driven architectures are an ideal way to communicate across business domains or with 3rd party systems. An example analogous to the order fulfillment example earlier could be the cross-selling of a new financial product to an existing customer. For many financial organizations, this might involve customer validation, screening from a 3rd party service, security or fraud checks, creation of a new customer account for the new product, and an update to a central customer profile service. Instead of a large batch-driven process, this could instead be managed gracefully as a series of asynchronous business requests, representing notifications to other systems and acknowledgements back to those systems that the event has been processed.

Fraud Detection

As the sophistication of fraud continues to evolve, so do the tools used to detect fraudulent activity. Fraud and other malicious activity detection are frequently implemented as event-based systems. Event-based systems can both allow for quick detection of malicious activity while also handling bursty load, without blocking high-volume transactions.

Streaming Analytics

Fraud detection is a specific case that can be generalized broadly under the category of streaming analytics, which run well under an event-driven architecture. Streaming analytical systems can completely decouple the analytical use cases from the transactional processing, allowing for powerful and immediate insights from activity streams with no impact to the transaction node. Streaming analytics use cases such as cross-selling and promotional offerings can be generated in real-time and therefore generate far better customer engagement.

Cautions Around Real-Time Systems

These near-time, event-driven systems are great, but require caution and care in their implementation. For example, many batch processes (for example, those using Spring Batch) depend on a job composed of serialized steps. This straightforward approach doesn't apply as well to real-time systems, where multiple streams of inter-related events are being processed concurrently. To do lookups or aggregations on these constant streams of data typically requires some sort of materialized view of the event stream. Taking the trading example we have used several times, a quote microservice may utilize, for example, an in-memory data store that maintains current price levels. This data store allows other systems calling the microservice to see the current price of any asset, instead of consuming events to understand the current quote.

In addition, error handling changes dramatically. In synchronous systems, calling an API returns an immediate error; in event-driven systems, the consumer of a message would, upon encountering an error, post to an error queue that all interested parties must subscribe to. Not only is this error handling an extra step, but it's done outside of prior processing flow. This often means that any error handler needs to do compensating activities instead of simply throwing an error. And even though an event-driven system in many respects automatically audits and logs itself, diagnosing errors can be more difficult. Developers need to navigate a series of event queues and processing services to understand where and why an error occurred.

Finally, event-driven architectures are a serious mindshift for development and operations teams. Developers require careful consideration of good event-driven architecture standards - what constitutes events vs. messages, event store models, stream processing libraries and CQRS patterns should all be evaluated by teams before starting development. Current research suggests that developers and designers who are ill-equipped for this transition typically revert to basic API and one-to-one notification relationships. In addition, selecting the right tooling is vital - Kafka, RabbitMQ, and cloud message buses all merit consideration given the right use cases. Operators and administrators must be prepared to manage and administer message-based infrastructures and event services. Security, scalability, redundancy and audit must also be considered.

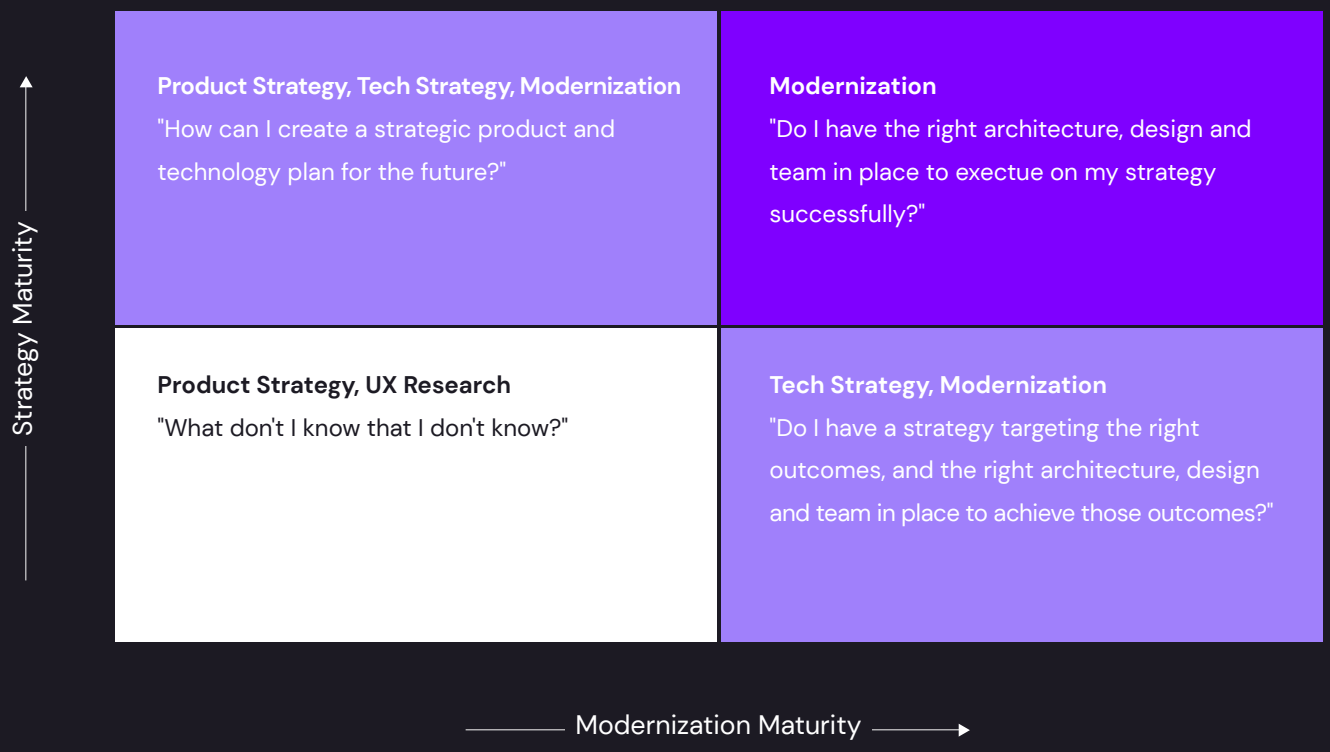
Where to Start?

Our proposition

We have found firms are most successful with their modernization efforts if they assess their business in terms of Strategy Maturity and Modernization Maturity. The matrix below will help answer where you should begin your efforts, based on which quadrant your business falls into. For example, if you firm is high in Strategy Maturity but low in Modernization Maturity, you will benefit from engagements focused on Product Strategy, Tech Strategy, and Modernization.

Where Do You Start?

Assess your strategy maturity and modernization maturity to determine where to start.



I need to	How to get started	The value we deliver
Identify and prioritize specific opportunities that real-time processing creates for my business	**UNDERSTAND** Competitor Audit / Landscape Analysis to identify where your competitors are leap-frogging you, and where your tech is hindering your business. **DISCOVER** Conduct user research and experience mapping to identify opportunities where your customers or partners are feeling pain. Conduct workshops with stakeholders and frontline staff to generate more ideas. Put some of your ideas in front of your customers/partners, and co-create ideas with them. **DEFINE** Use what you've learned in your research and experimentation to prioritize which opportunities will have the most impact to your business, are aligned with other business goals, and have the fastest time-to-value.	**MODERNIZATION STRATEGY SPRINT** Our strategic product and technical teams work with you collaboratively to understand your current business and technology landscape, understand the competition, customers and partners. Then we guide technology decisions based on the opportunities we identify and prioritize with you.
Modernize the systems that I've already prioritized.	**APPLICATION ARCHITECTURE** Perform just enough to get started Deep technical review Event storming and domain-driven design Sprint 0 Sprint DevOps	**APPLICATION TRANSFORMATION ACCELERATOR** We have extensive experience building new services from the ground up that return 10x or more value for your business.

Financial Services

Conclusion

At Kin + Carta we believe moving from batch to real time technology is critical for enterprise financial institutions. Any investment in moving from batch to real-time systems must be considered within a broader modernization strategy focused on people, process and technology. By starting with a holistic understanding of your business, we're able to build new foundational infrastructure that leverages existing legacy systems, while not interrupting them. The ferries in Brooklyn didn't stop running during the construction of the Brooklyn Bridge. In fact, they're still running today. Those ferries are just serving a much more prosperous city than if they were still the only way to cross the Hudson.

Get in touch today, we'd love to talk further about what real-time can mean for your business.

Appendix

https://www.cnbc.com/2019/04/26/amazons-free-one-day-shipping-puts-the-pressure-on-walmart-target.html

https://www.cnbc.com/2019/05/13/walmart-announces-next-day-delivery-firing-back-at-amazon.html

https://c1.sfdcstatic.com/content/dam/web/en_us/www/assets/pdf/salesforce-state-of-the-connected-customer-report-2019.pdf

https://www.nerdwallet.com/blog/credit-cards/instant-credit-card-numbers/

https://www.zellepay.com/press-releases/zelle-study-finds-growing-use-of-digital-payments-across-generations

https://www.businessinsider.com/google-will-begin-offering-checking-accounts-2019-11

https://www.statista.com/statistics/881033/call-center-market-size-region/

https://www.gartner.com/document/3956041?ref=solrAll&refval=241185750

https://www.gartner.com/document/3942102?ref=solrAll&refval=241181344

https://www.gartner.com/document/code/352548?ref=authbody&refval=3956041

Authors

Eric Rosenzweig,
MD Financial Services

Joe Nedumgottil,
Senior Technical Principal

Mark Ardito,
VP of Cloud Modernization

Jared Johnson,
Sr. Principal Digital Strategist

Gerry Koci,
Enterprise Architect

Get in touch.

Mark Ardito

VP of Cloud Modernization
mark.ardito@kinandcarta.com

T: +1 (312)479 2108

Global Headquarters

1 Tudor Street
London
EC4Y 0AH
United Kingdom

T: +44 (0)207 928 8844

US Headquarters

7th Floor, 111 N. Canal St
IL, Chicago
60606
United States

T: +1 (866) 380 8472