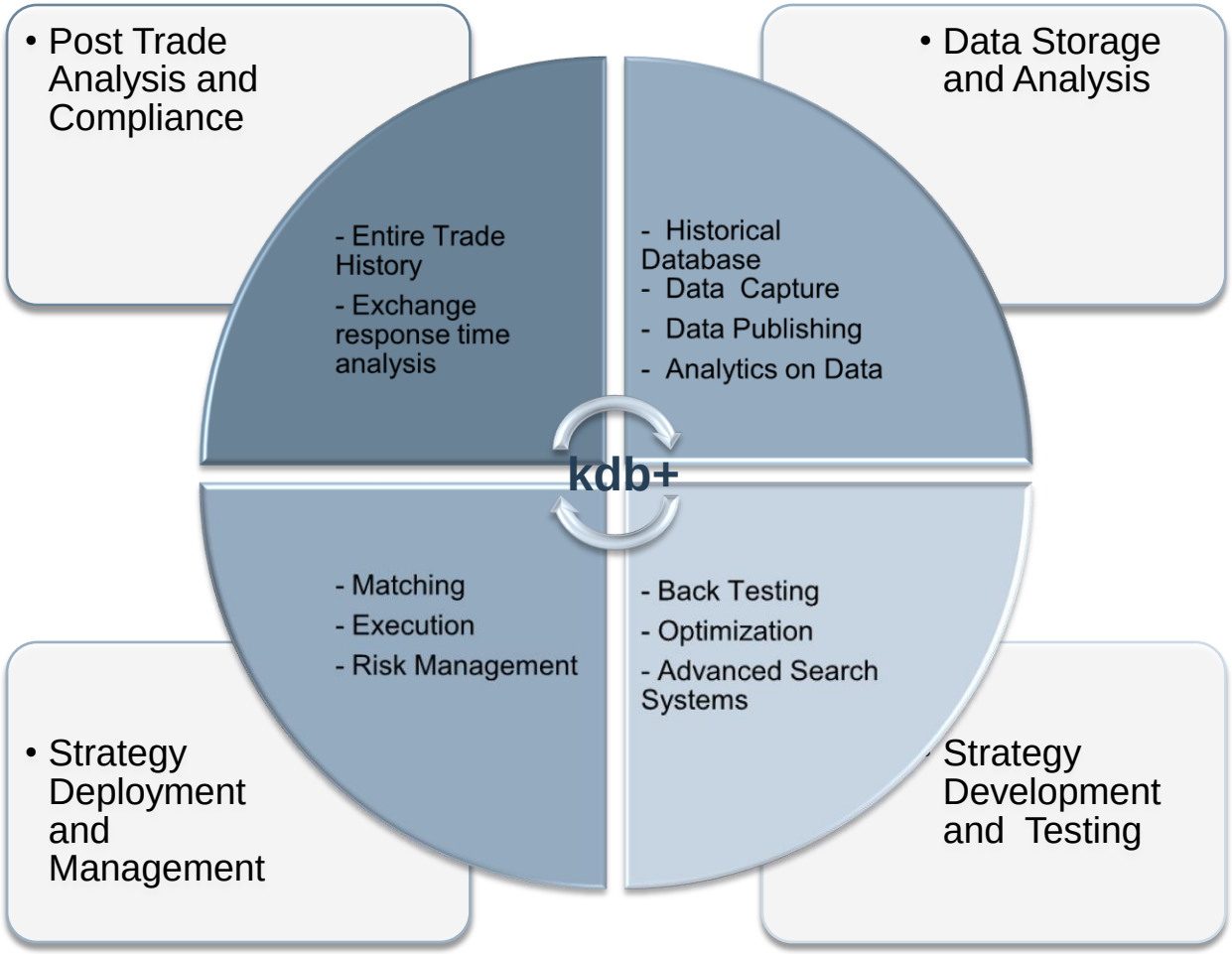
AUTOMAGICAL:
BUILDING FULLY AUTOMATED TRADING SYSTEMS IN KDB+

Jacob Loveless
BGC Partners

Overview

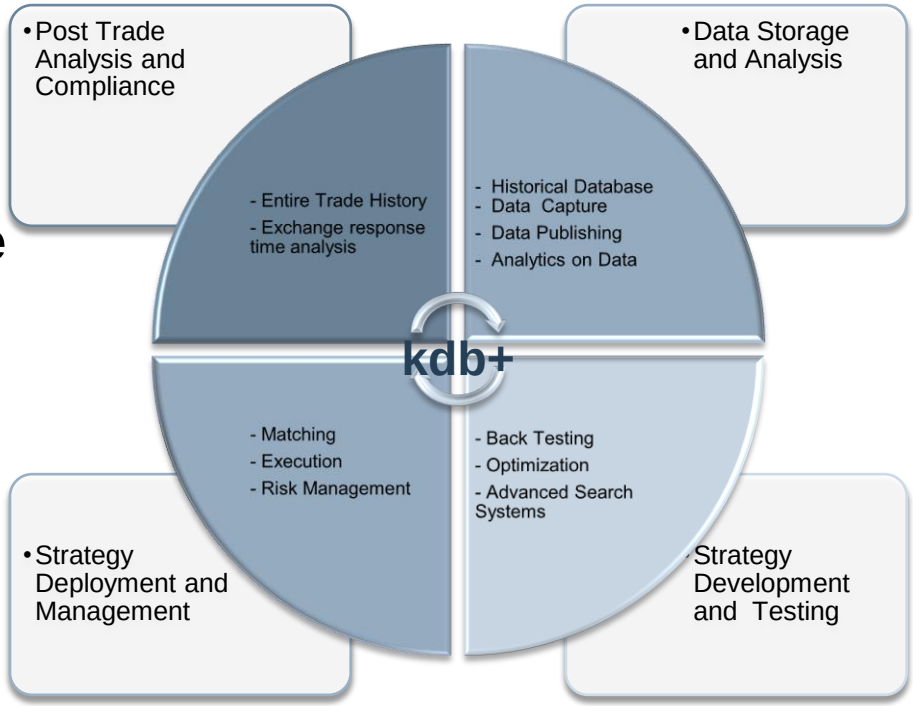
- Algorithmic Trading Process
- Algo Trading Architechitures
  - Component Interactions and Pitfalls
  - Filters, Algorithms and Models
- Next Generation Systems
  - Beyond Colocation: working with your exchange
  - Order flows
  - Beyond level II analysis
- Conclusion

Algorithmic Trading Process



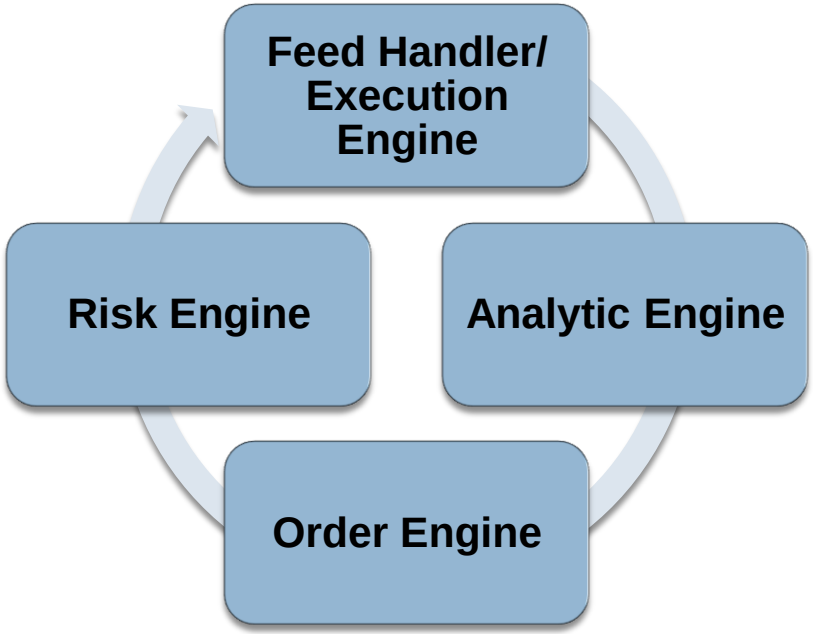
Algorithmic Trading Process

- In my experience, the more homogenous the system, the better.
- It is important that your historical data schemas match your realtime schemas.
- Historical analytics need to match your realtime analytics.
- **Intergration is unnecessary when everything is the same**



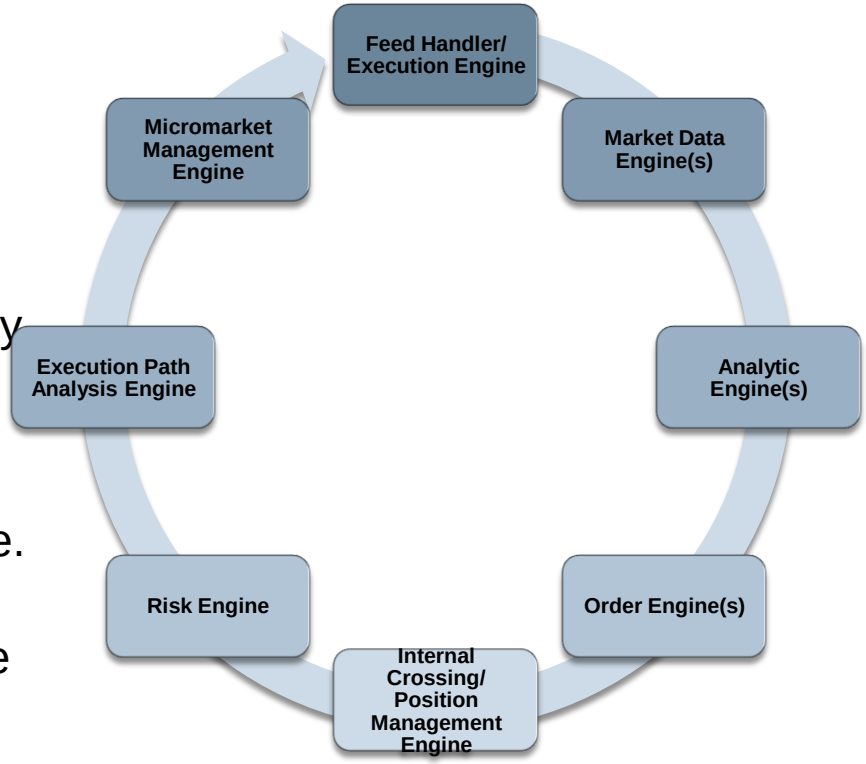
Algo Trading Architechitures

- The basic Architechture is:
  - **Feedhandler**: obtain data
  - **Analytic Engine:** perform calculations
  - **Order Engine:** check trading rules
  - **Risk Engine:** confirm execution request
  - **Execution Engine:** execute

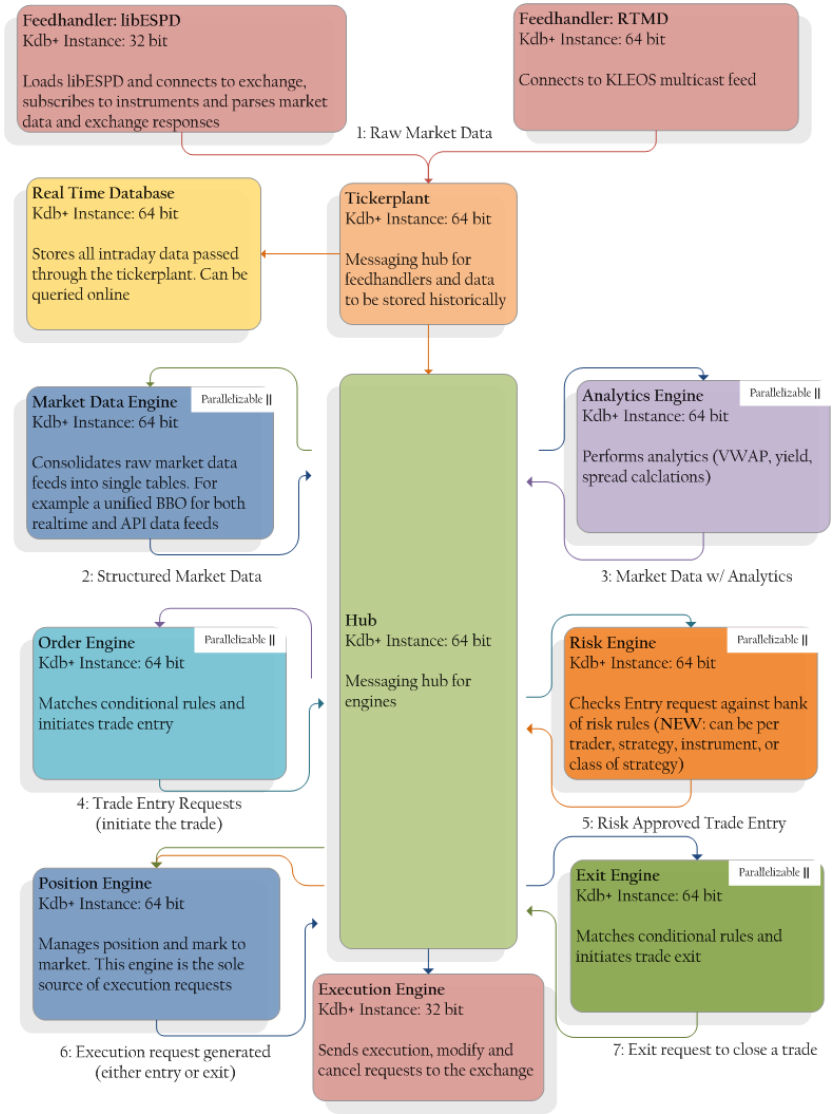


Algo Trading Architechitures

- … but everyone's is a little different
  - **Incorporation of flow desk**
  - **Cross market trading**
- The more communication points, the more the need for efficent communication.
- Race conditions ruin the data. The only thing worse than no data at all, is untrustworthy data
- In general, the process should be as complicated as necessay- but no more.
- **Simplify**: Fools ignore complexity, pragmatists suffer if. Some can avoid it, genius removes it.

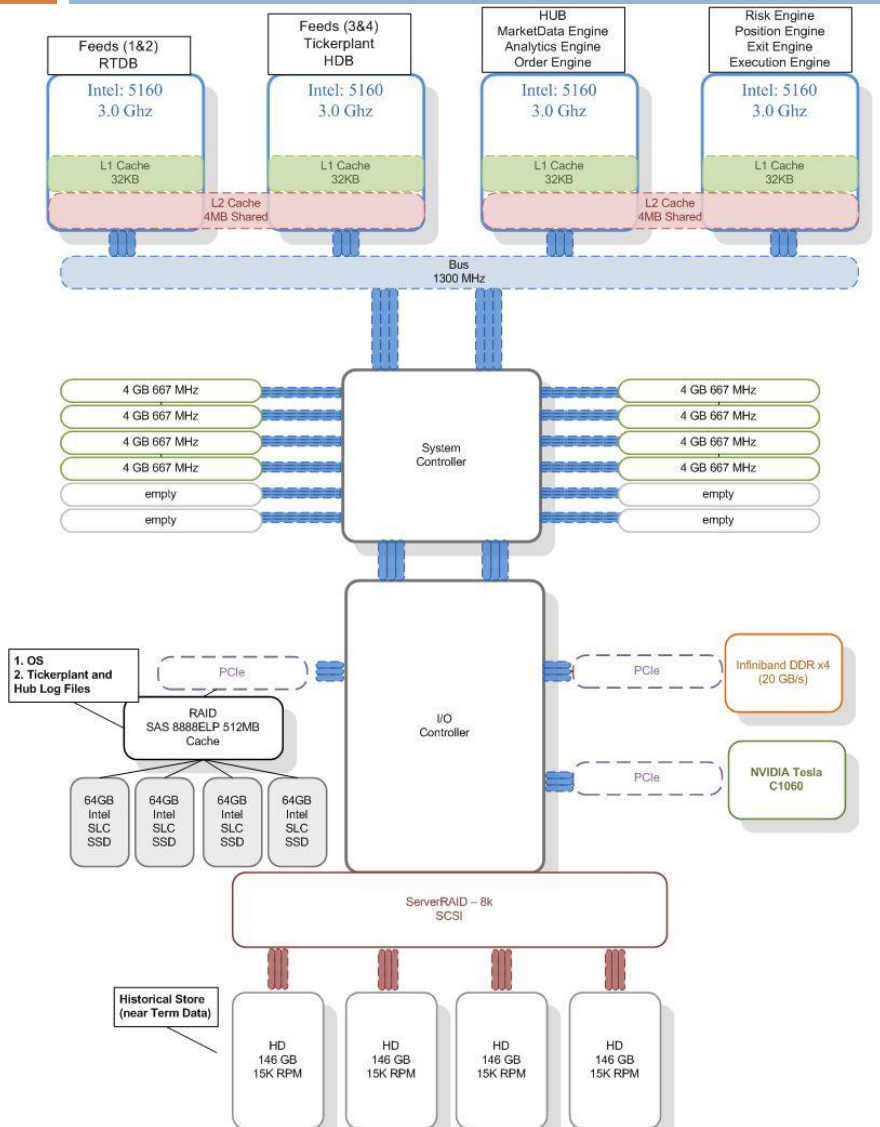


Algo Trading Architechitures



- This is an example of a production architecture (FD Delta).
- End to End latency is ~200 us
- Used to trade multiple markets (UST, IRF,FXF,FX Spot) via BGC exchange.
- Uses multicast collocation feed and API feed.

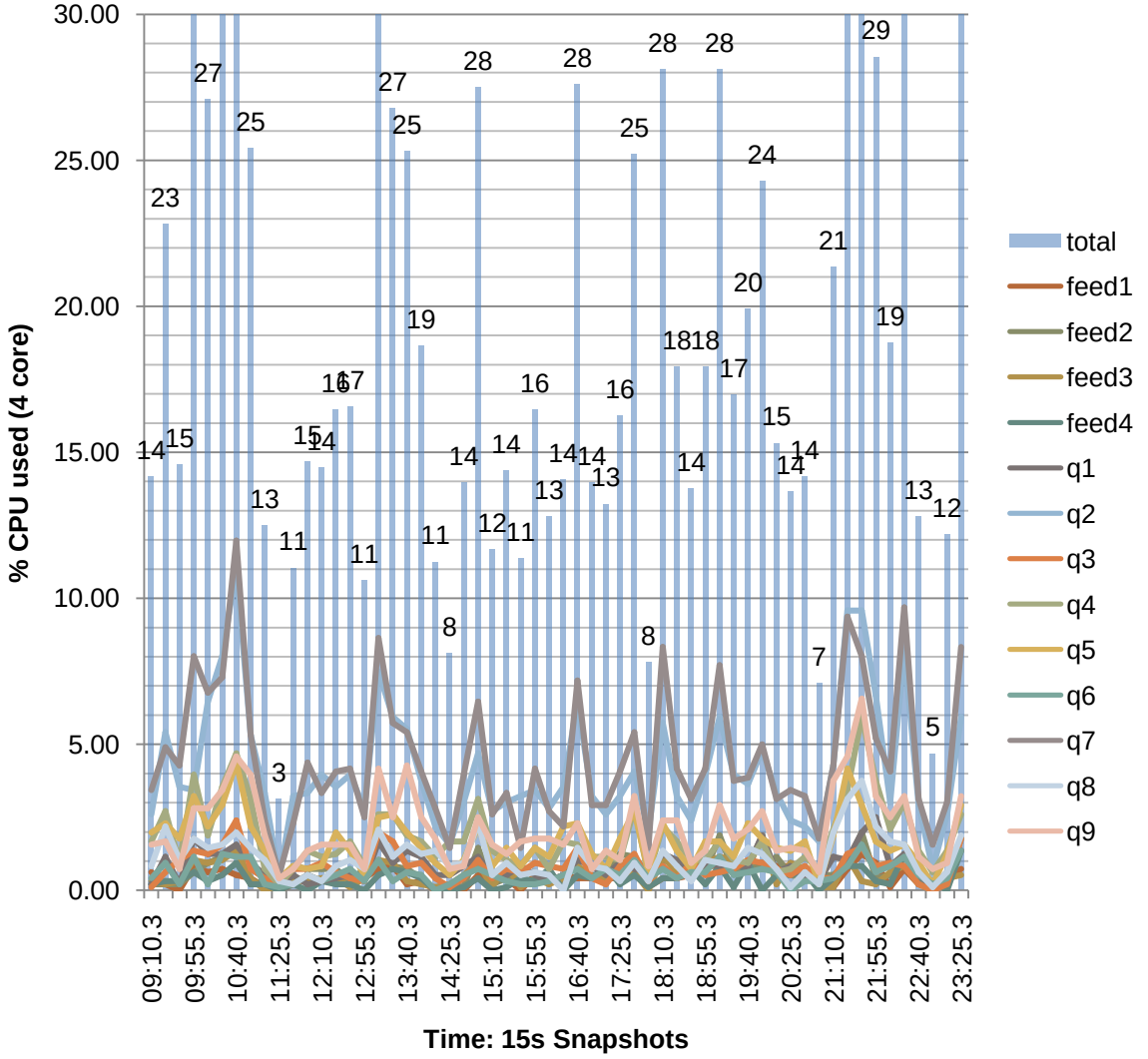
Algo Trading Architechitures



- Little Data Center in a box.
- SSDs are nice but not necessary. Disks are of little concern (in memory all day).
- Infiniband is necessary for any DR
- Pin Process to CPU in order of workload
- Linux is better then Windows
- Real time Linux is better than Linux
- Intel 5160 was easy to Overclock- and stable.

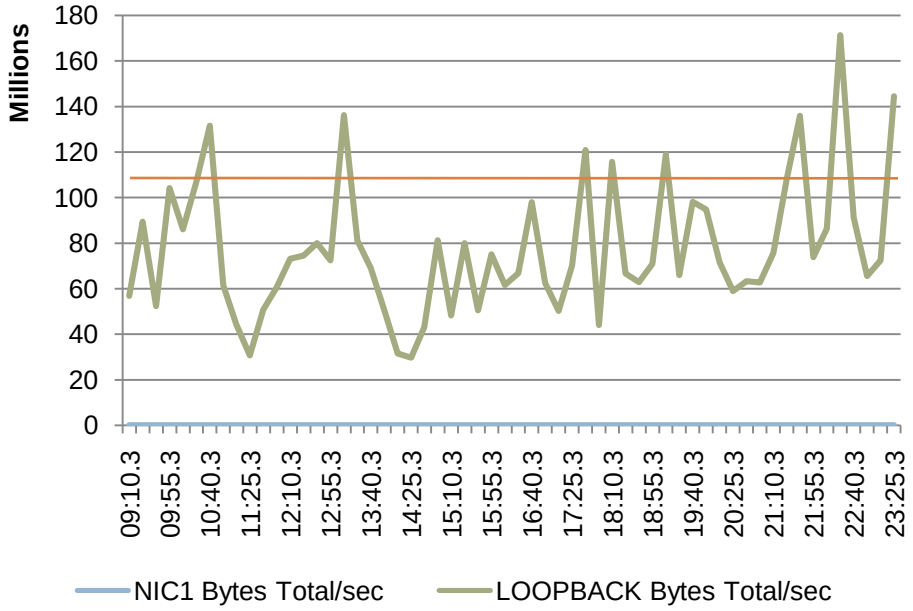
Algo Trading Architechitures

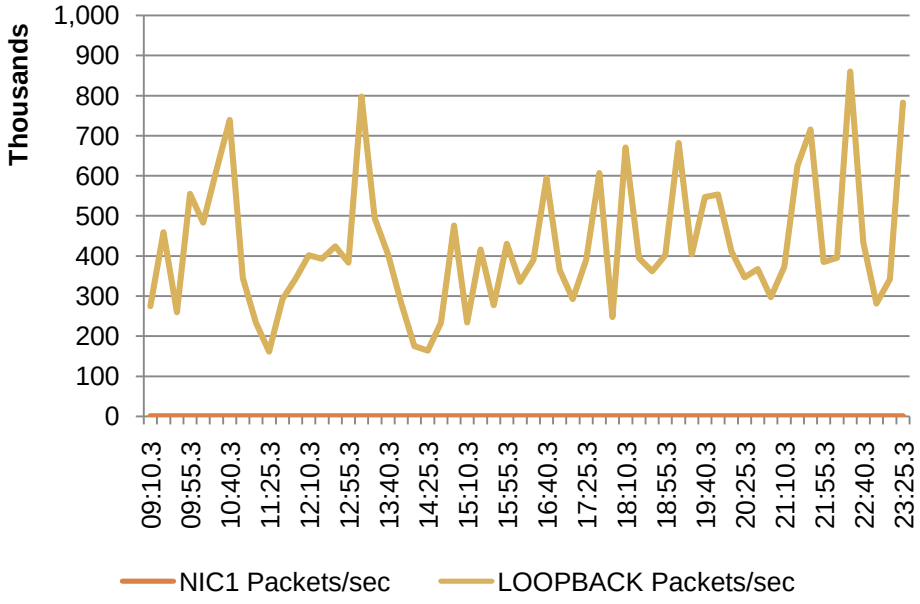
- Multi-process model (still)
- Very difficult to find use case for multithread
- Pin process to CPU according to usage
- Set as priority process
- Disable paging
- **Multi-process allows for simple horizontal scaling in theory…**



Algo Trading Architechitures

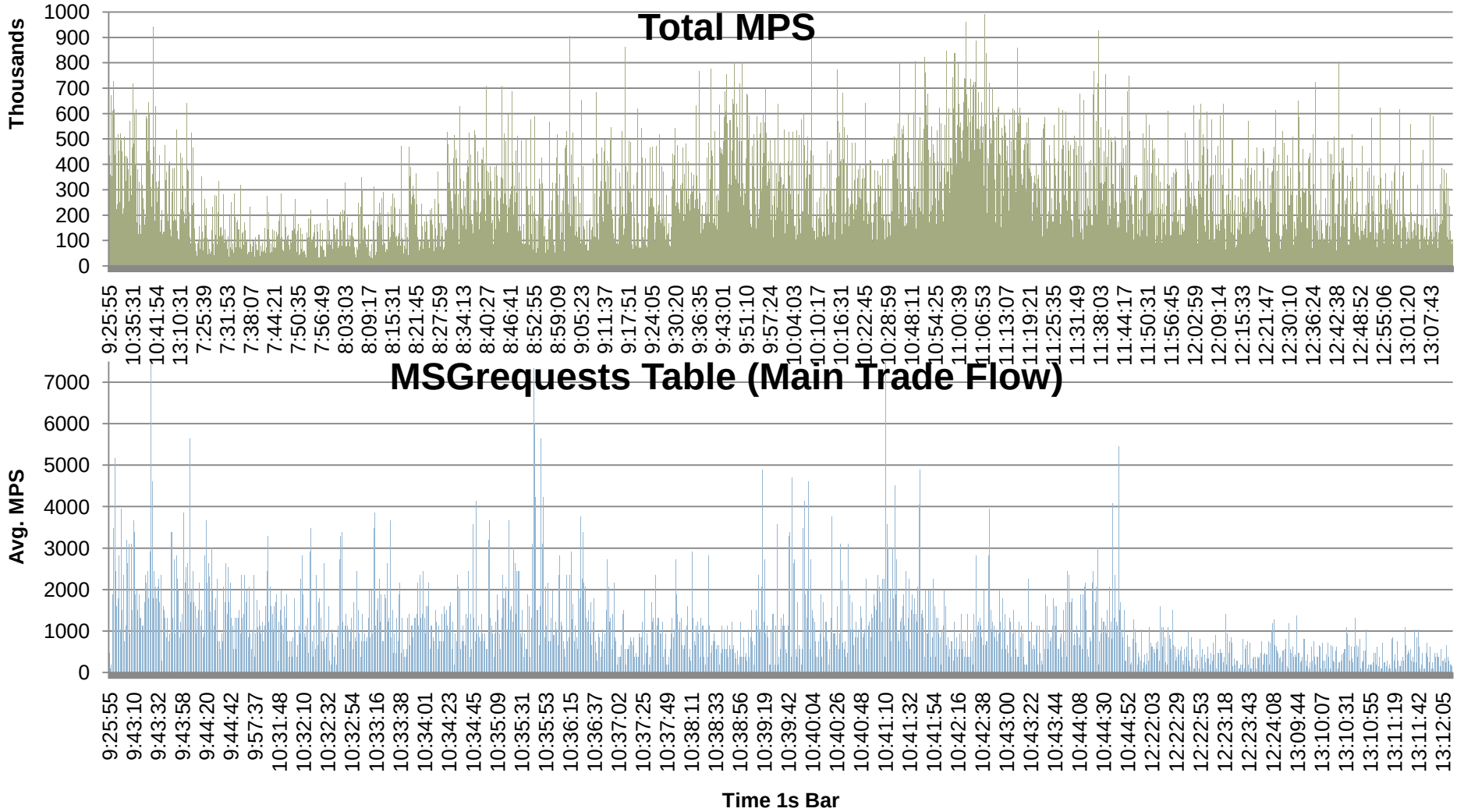
- **… but in practice IPC is significant**
- Infiniband helps here.
- Shared memory would be awfully nice (Arthur?).
- DR looks easy on paper, but it's expensive in practice. A chained hub will exhaust a 100MB connection.
- Have to find the balance of CPU Utilization vs.. Network Load for multi-server deployment.
- It's very nice having it all on one box. But can you sleep at night?





Algo Trading Architechitures

Bumping up against 1mm /sec mark (scary). It's hard to partition the messages (cross asset)



Algo Trading Architechitures: Pitfalls and Failures

- Model Synchronization Issues
  - **Calculation Engines:** Back testing systems utilize different code than online systems
  - **Order Engines:** Assumptions are made about the speed at which models can be matched to data
  - **Execution Engines:** Assumptions are made about the speed execution.

Algo Trading Architechitures: Pitfalls and Failures

- Model Synchronization Solutions
  - **Calculation Engines:**
    - Utilize the same code as backtesting systems. Either backtest (play forward data), or share the same analytics.
    - Be sure you can build your exact data on the wire (filter bad ticks?)
  - **Order Engines:**
    - Parallize the matching process when possible.
    - Separate the matching process from the calculation process (e.g. simple matches rules based on complex calculated values).
    - Understand the inherent latency of your systems.
  - **Execution Engines:**
    - Separate the execution process.
    - Store your trades to develop estimates of market impact, slippage etc.
    - Understand your model interactions (can you cross internally?)

Algo Trading Architechitures: Pitfalls and Failures

- Market Synchronization Issues
  - **Calculation Engines:** Focus has been on getting the data to the decision process as quickly as possible. From there the process is often sequential. How long does it take to react to the data?
    - Example: Low latency market data feed returns prices across the futures curve. The model receives the prices and calculates the “true value”, and requests a trade for the dislocated instruments. As volatility increased, these calculations took longer- and the system was pricing off market.
  - **Execution Engines:** Who sets the price? If the price is set at the order engines (trade decision engines), will the market have moved before returning to the wire?
    - Example: A high frequency stat arb system set the price at the order engine level. As market prices changed quickly (often as a result of illiquidity), the HFT model began to place bids at offer levels (thus cross spread).

Algo Trading Architechitures: Pitfalls and Failures

- Market Synchronization in **General**:
  - Be careful about engines which go “message crazy”
  - It’s very difficult to manage the message queue. We set timestamps at the engine level, and “throw away” messages outside of a threshold for certain tables. Coding feedback and reaction is hard (and costly).
- Market Synchronization Solutions for **Calculation Engines**
  - Benchmark your calculation engines. Calculation engines should be able to perform at 2-3x the feedhandler capacity.
  - Along the same lines, use incremental calculations when possible. When in doubt, see if a faster calculation is available during peak load (e.g. switch from correlation to an FFT implementation). Always see if you can pre-calculate (e.g. yield lookup tables). If you pre-calculate, pre-calculate 3x more than you think you will ever need (memory is cheap).
  - Ask yourself if you are willing to give up some degree of accuracy for speed. This is a big problem with pre-packaged calculation frameworks.
  - Obviously all the standard coding optimizations apply.
- Market Synchronization Solutions for **Execution Engines**
  - Set the price at the calculation level, but allow for a degree of error. Ideally your models have a level freedom inherently.
  - **Seperate your micro-market execution from your model!** This is a simple solution to most problems. The rules and techniques behind the micro-market are generally not specific to types of trades. Break out your actual execution process from your model.
  - At the very least, have a set of basic checks at the last stage (current market vs.. request, instantaneous spread etc.)

Expect the worst….

- Expect, and model to the worst possible scenarios (from the market)
  - GHPT
  - Introduce non cooperative noise into your data
  - Introduce bad ticks into your data
  - It is arguably as important to develop “meta models”, models which describe when models aren’t applicable as the actual trading models.
    - For example: News events are difficult to code. Develop methods for detecting illiquidity.

This Market:

sym	pos	time	bid	bidsize	bidlot	ask	asksize	asklot
usg_10Y	0	12:29:53.797	102.1875	7	1 1 3 2	102.2031	8	5 1 1 1
usg_10Y	1	12:29:54.141	102.1719	9	2 1 1 1 1 2 1	102.2188	7	1 1 1 2 1 1
usg_10Y	2	12:29:54.141	102.1563	8	2 1 2 1 1 1	102.2344	8	1 1 1 1 2 2
usg_10Y	3	12:29:54.360	102.1406	6	2 1 1 1 1	102.25	11	2 1 1 2 1 1 1 2
usg_10Y	4	12:29:54.141	102.125	4	2 1 1	102.2656	3	2 1f

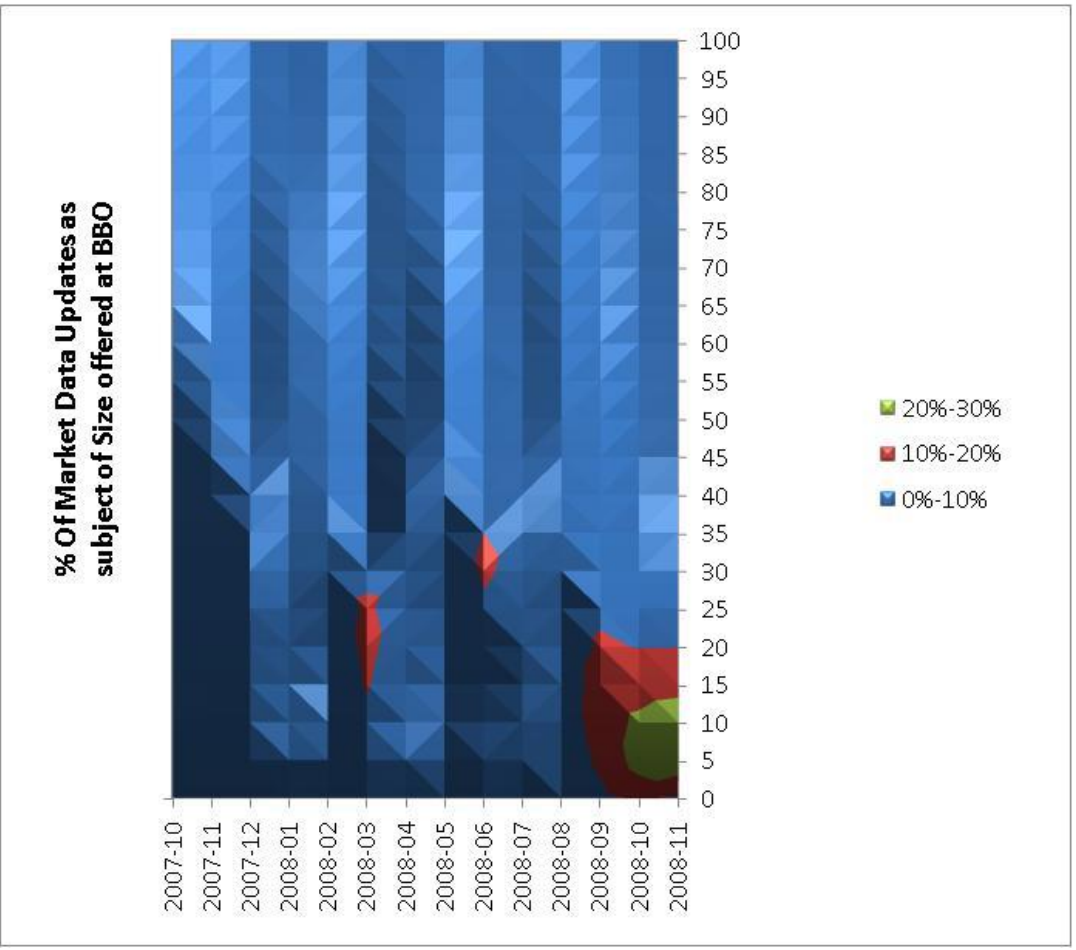
…is not the same as this market

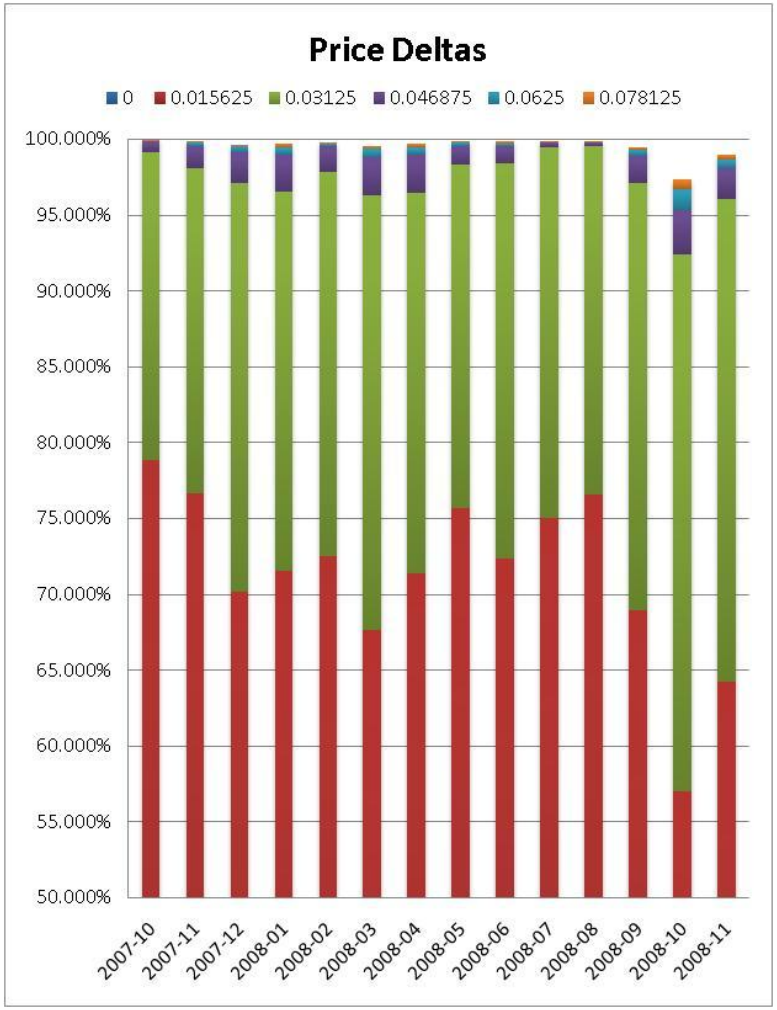
sym	pos	time	bid	bidsize	bidlot	ask	asksize	asklot
usg_10Y	0	08:29:58.394	102.625	2	2	102.7031	2	1 1
usg_10Y	1	08:29:56.769	102.5938	10	10	102.7188	1	,1
usg_10Y	2	08:29:56.769	102.5625	2	2	102.7344	1	,1
usg_10Y	3	08:29:50.410	102.5313	10	10	102.7969	1	,1
usg_10Y	4	08:29:57.222	102.4844	10	10	102.8125	1	,1

If you expect the worst …

… you'll be right one day

- Even in the US Treasury market, which is traditionally the most stable of the micro markets….
- **Uncertainty →Illiquidity → Volatility**





Algo Trading Architechitures: Filters, Algorithms and Models

- Filters: methods for smoothing or measuring the error rates of data
  - Simple: VWAP, TWAP, SMAVG
  - More Complex: Kalman, Unscented Kalman, Double Exponential Filtering
  - Rocket Science: Stochastic filtering (fitting data to your model), evolutionary adaptive filtering, particle filters…
- Algorithms: methods for measuring and calculating values.
  - Simple: Correlation, Covariance, Cointergration
  - Advanced: (trade secrets)
- Models: The combination of filters and algorithms to measure and estimate data

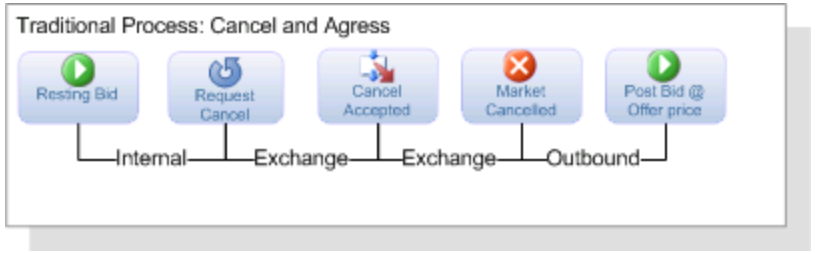
Algo Trading Architechitures: Filters, Algorithms and Models

- Nosce te ipsum. This is a major source of error
- **Bad**: Using filters &| Algorithms you don't fully understand
  - How sensitive are they to noise?
  - If you combine them, what is the result of the superposition? Independent? Constructive? Destructive?
  - What are the limiting behaviors of the functions? As variance increases, does computation time increase exponentionally?
  - What are the "perfect storms" for error.
  - If you understand your algorithms, you understand where they can be substituted.

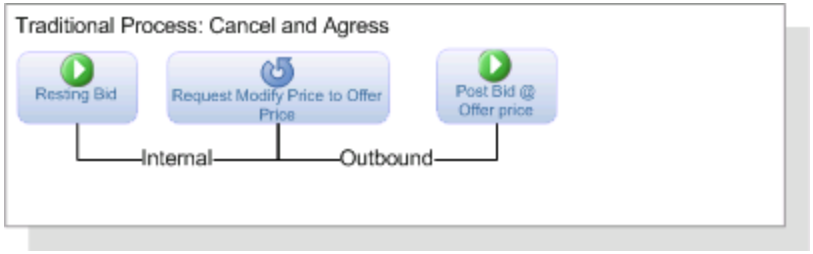
  <u>If you buy them, understand them (have the source code). If you build them, check them.</u>
- **Crazy**: Using Models you don't fully understand.
  - I could give a list of firms and examples. Ensure you understand all the implications of your models.
  - Develop "meta models" when possible. Understand how your models effect and are effected by one another.

Next Generation Systems

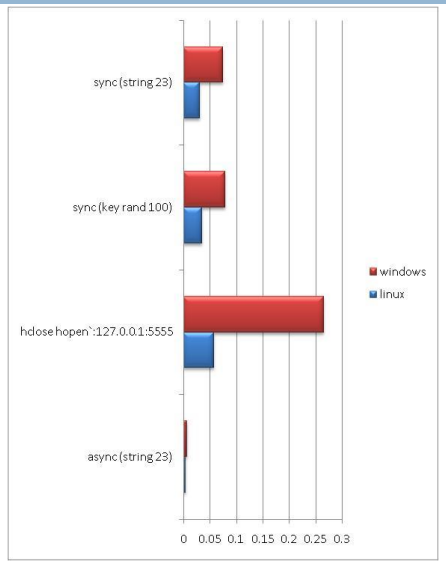
- Most firms are using collocation services.
  - Next steps are to use internal data center feeds.
  - For example, multicast wire level market data feeds
  - Faster delivery and parsing
- Extending logic to include advanced order types offered by the exchange
  - For example “modify” order types (BGC FX/UST and ELX Futures)



  - Requires wait in internal states (I must wait for the cancel). Instead use a modify order type.



Next Generation Systems



Real Time Kernels:

Almost a no-brainer. Noticeable difference (especially TCP and IO). Mature and supported (Redhat MRG and Suse)

Solid State Drives:

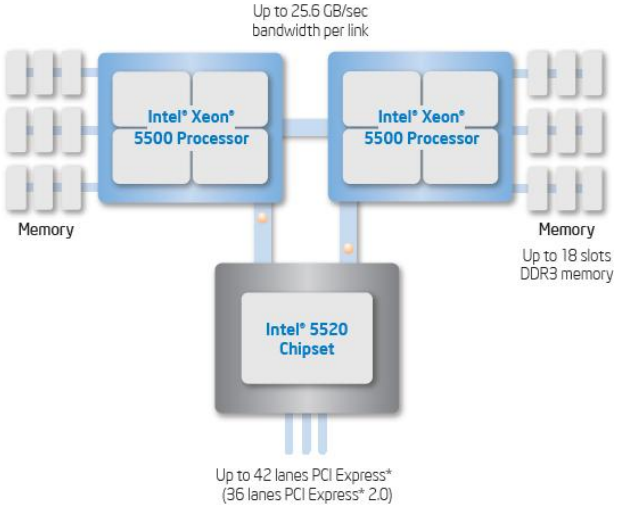
One day. Still too expensive. No real application in real time trading (everything is in memory, 1TB memory systems available).

New Intel and Multicore:

Makes sense in our world- at high clock speed (3.4 GHz).
Biggest benefit is DDR3 and QuickPath
Overclocking? Looks promising (5160 was very nice)

NVIDIA CUDA

Tesla is, simply, amazing.
But it's limited (4 GB memory).
It's hard to find embarrassingly parallel problems which warrant it.



Questions?