SESSION 5:
INTRODUCTION TO THE FINANCIAL SOFTWARE ARCHITECTURE AND COMMUNICATION

Sebastien Donadio
sdonadio@uchicago.edu

MSCA 32016 – Advanced Python for Streaming Analytics

Agenda

- JSON
- Finance and Analytics

JSON Message

https://docs.python.org/2/library/json.html

What are we trying to solve?

- How to send a class from a program to another one?

- Example:

```
class orderbook:
   def __init__(self):
      self.length = 20
      self.bestOffer=10
      self.bestBid=11
```

1st method

- We convert the class to a dictionary and send the dictionary

```
a=orderbook()
s=a.__dict__
send(s)
```

When decoding, we can use another class

```
ast.literal_eval(s)
```

Specfic to Python

2nd method: Marshalling/Pickle

```
class orderbook:
  def __init__(self):
    self.length = 20
    self.bestOffer=10
    self.bestBid=11


a=orderbook()

s=marshal.dumps(a.__di
ct__)
print s

b=marshal.loads(s)

print b
```

```
class orderbook:
  def __init__(self):
    self.length = 20
    self.bestOffer=10
    self.bestBid=11


a=orderbook()

s=pickle.dumps(a)
print s

b=pickle.loads(s)
print b
```

2nd method: Marshalling/Pickle

- **marshal** cannot be used to serialize user-defined classes and their instances.
- The **marshal** serialization format is not guaranteed to be portable across Python versions.
- The **pickle** module keeps track of the objects it has already serialized, so that later references to the same object won't be serialized again. marshal doesn't do this.

3rd method: JSON protocol

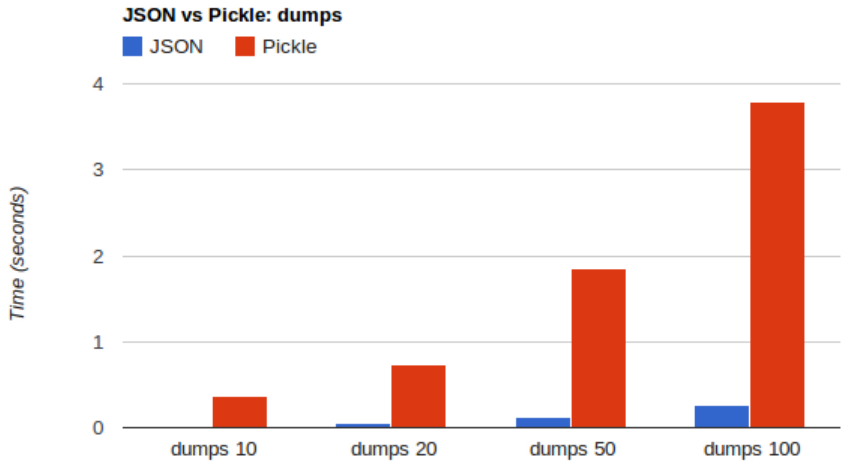
```
import json

class orderbook:
  def __init__(self):
    self.length = 20
    self.bestOffer=10
    self.bestBid=11


a=orderbook()

s=json.dumps(a.__dict__)
print s

b=json.loads(s)

print b
```

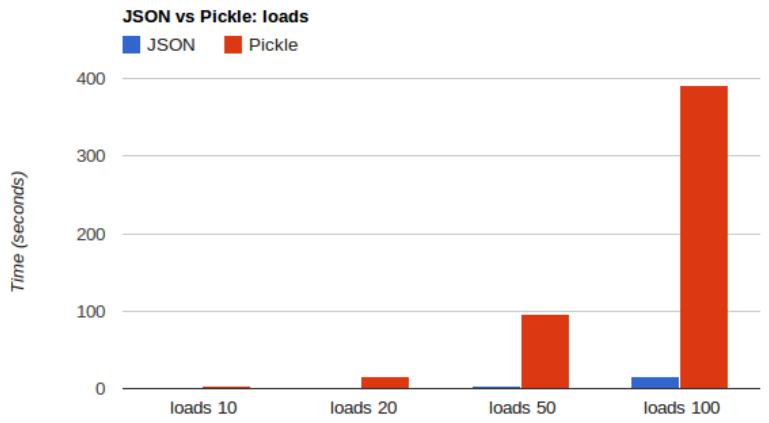
JSON

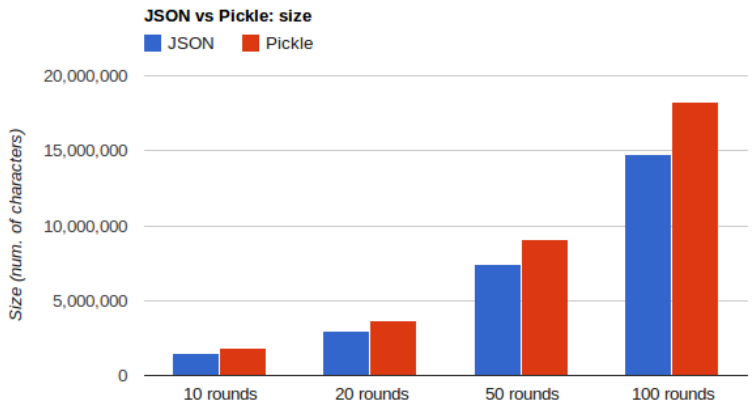
lightweight format that is used for data interchanging

- JSON is built on two structures:
  - A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.
  - An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

JSON speed


JSON vs Pickle: dumps
JSON
Pickle
Time (seconds)
4
3
2
1
0
dumps 10
dumps 20
dumps 50
dumps 100


JSON vs Pickle: loads
JSON
Pickle
Time (seconds)
400
300
200
100
0
loads 10
loads 20
loads 50
loads 100


JSON vs Pickle: size
JSON
Pickle
Size (num. of characters)
20,000,000
15,000,000
10,000,000
5,000,000
0
10 rounds
20 rounds
50 rounds
100 rounds

JSON messages - encoding

```
import json
order={'price': 12, 'volume': 100}
a=json.dumps(order)
Out[]: '{"volume": 100, "price": 12}'

print (json.dumps(order, sort_keys=True,
indent=4, separators=(',', ': ')))
{
    "price": 12,
    "volume": 100
}
```

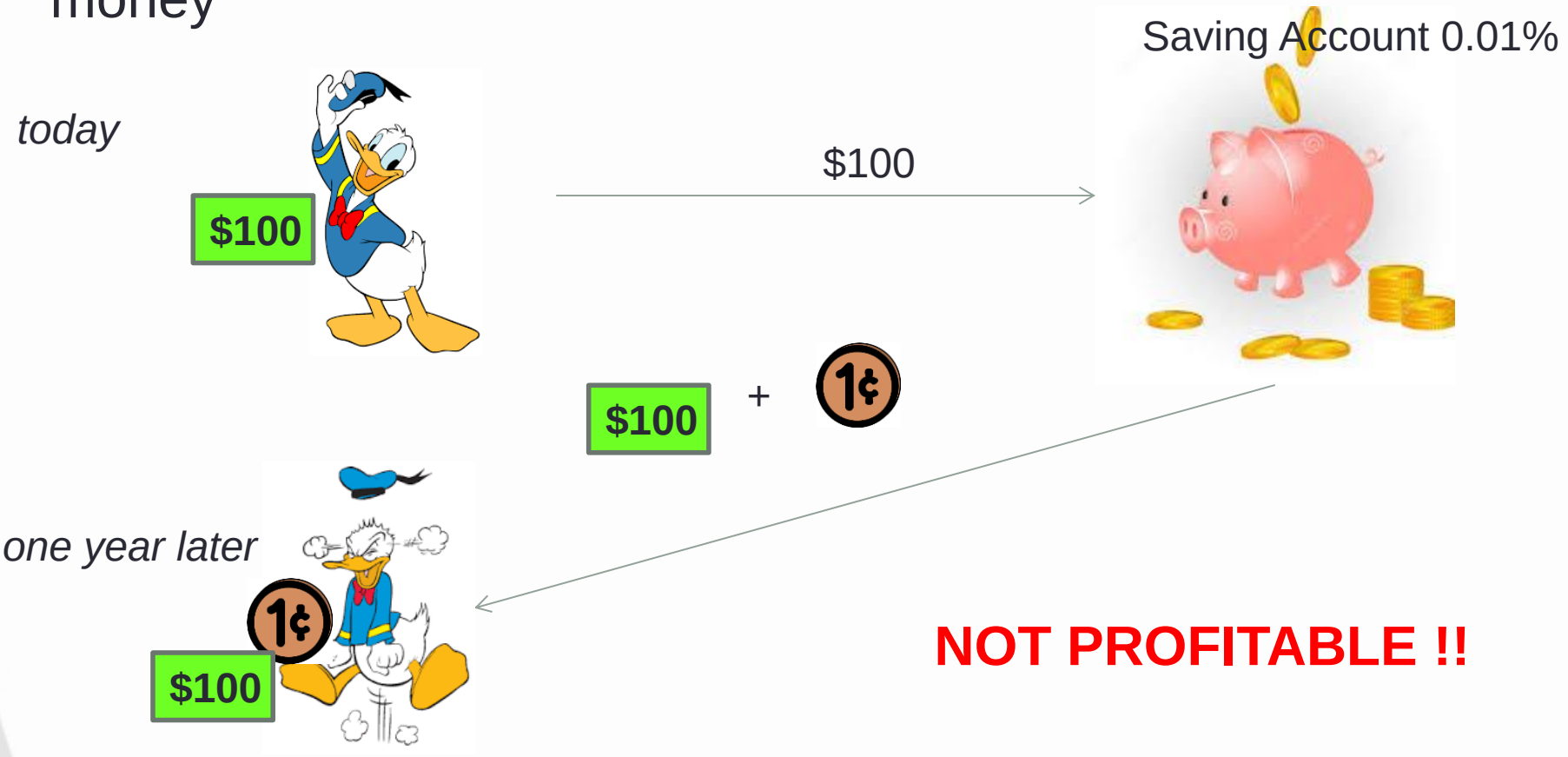
JSON messages - decoding

```
import json
json.loads(a)
{u'price': 12, u'volume': 100}
```

Software Architecture for a Trading System

Motivation (Saving Account)

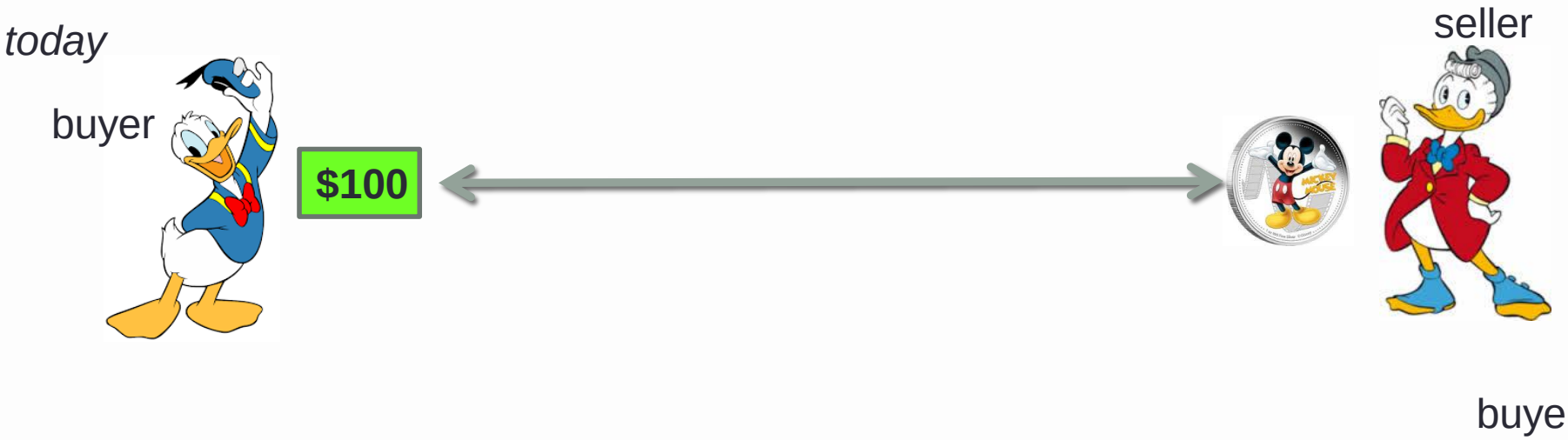
- Suppose you have $100 and you want to invest your money



NOT PROFITABLE !!

Motivation (Long term investment)

- Let's invest in something which has a better return than saving accounts (Mickey Quarter)



Motivation (Long term investment)

- Let's invest in something which has a better return than saving accounts (Mickey Quarter)

today

buyer

seller

$100

6 month later: MQ appreciates by 10% (because of shortage)

$110

buyer

Better but I need more money?

Motivation (Faster profit)

- Let's invest my money into many quarters

Motivation (Faster profit)

- Let's invest my money into many quarters

today

buyer

seller

$400 debt

lender

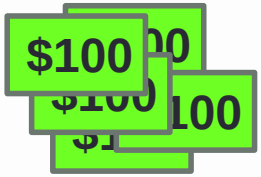
Motivation (Faster profit)

- Let's invest my money into many quarters

today

buyer

seller



$400 debt

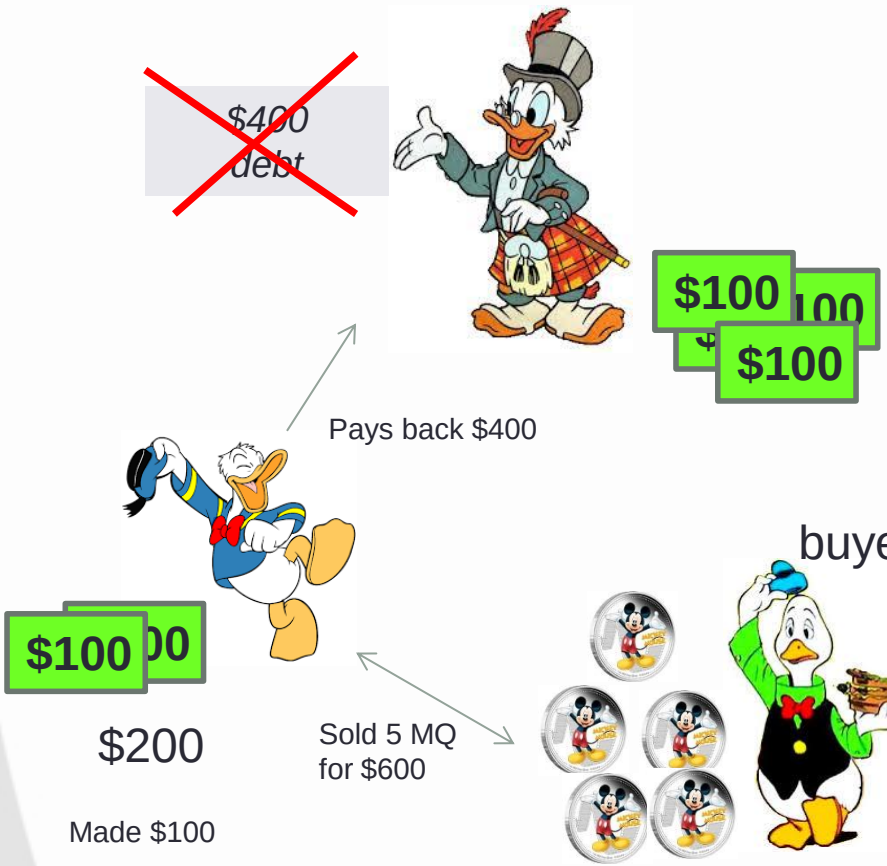
lender

Motivation (Faster profit)

Too much exposure
For a long period !

After 6 month

case 1: appreciation by 20%



case 2: depreciation by 50%

Motivation (Use a less risky solution)

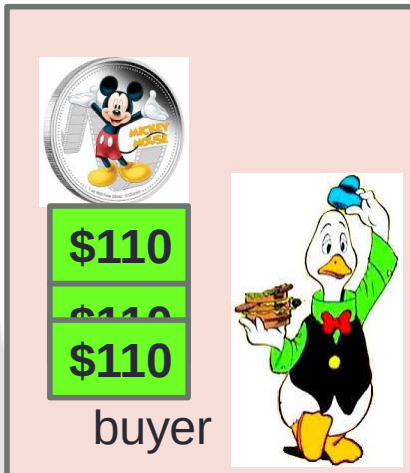
At a given time,
Donald already knows
that he can buy for $100
and sells for $110

Motivation (Use a less risky solution)

At a given time,
Donald already knows
that he can buy for $100
and sells for $110

Motivation (Use a less risky solution)

At a given time,
Donald already knows
that he can buy for $100
and sells for $110



Motivation (Use a less risky solution)

At a given time,
Donald already knows
that he can buy for $100
and sells for $140

Less risky !
If the price drops, Donald stops buying

MORE MONEY???
Of course Yes !

Motivation (Scale-up)

$100

$10*4

$20*4

buyer
seller

Wants to sell MQ
for $90

seller

London

Wants to sell MQ
for $100

seller

New York

buyer

Chicago

Wants to buy MQ
for $110

Limitations(Scale-up)

$100

$100

$10*4

faster

faster

$20*4

$100

Chicago

buyer

Wants to buy MQ
for $110

Limitation - Solutions

- Buy/Sell different products
- Buy/Sell different markets
- Being faster






MM $160

$100

buyer
seller


MM $150

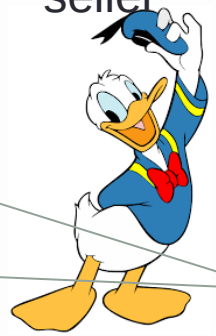


MQ $100


MM $130



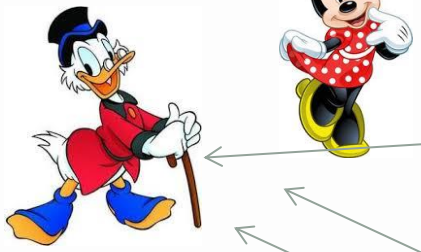
MQ $80



buyer

Limitation - Solutions

- Buy/Sell different products
- Buy/Sell different markets
- Being faster





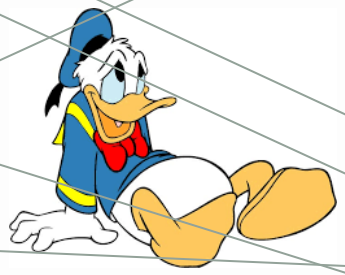

MM $160

MM $150

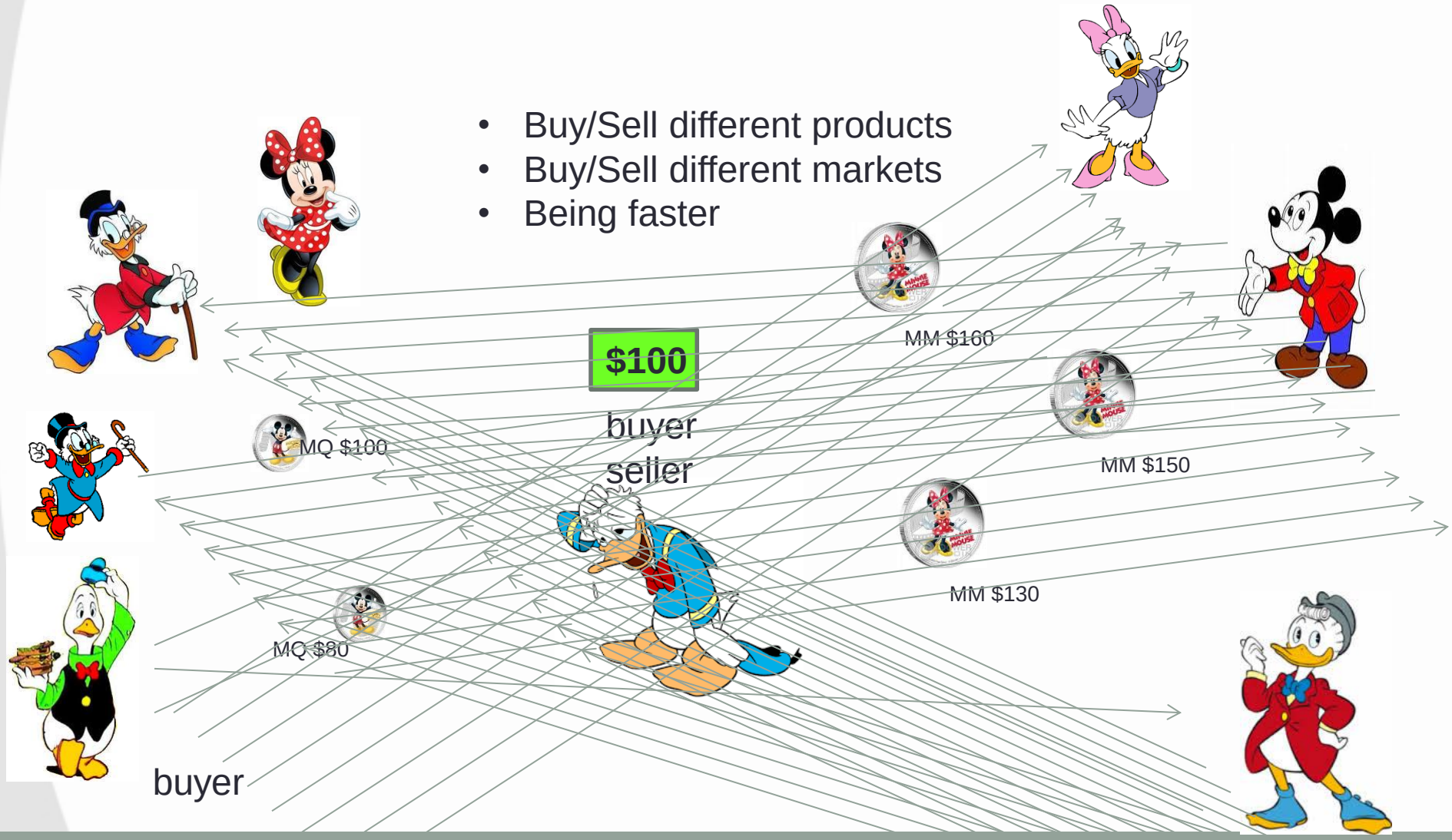
MM $130


MQ $100

MQ $80

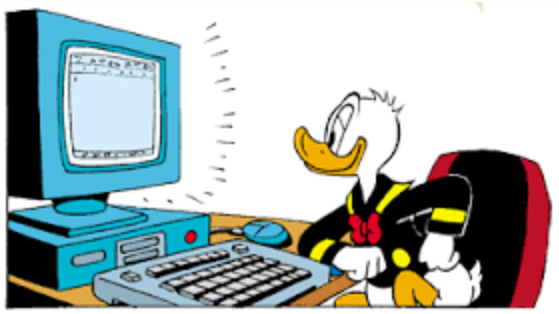
buyer
$100
buyer
seller



Limitation - Solutions



Motivation

- Need a system:
  - Reliable
  - Fast
  - Adaptable



We need a Trading System

Design – Communication

- Communication between the trading system and the venues (the exchange)
- Exchange specific protocol
- Electronic communication
- Goal: Converting the messages into internal structure of the system

Design – Book building

- Book creation
- Sorting by price, by quantity by venues

Venue	Vol	Bid	Offer	Vol	Venue
V1	1,000	1.21	1.31	5,000	V3
V3	1,500	1.20	1.32	2,000	V2
V2	2,000	1.20	1.32	1,500	V1
V5	5,000	1.20	1.32	1,500	V6
V7	1,000	1.19	1.33	1,500	V5

Design – Strategy

- Signal
- Execution

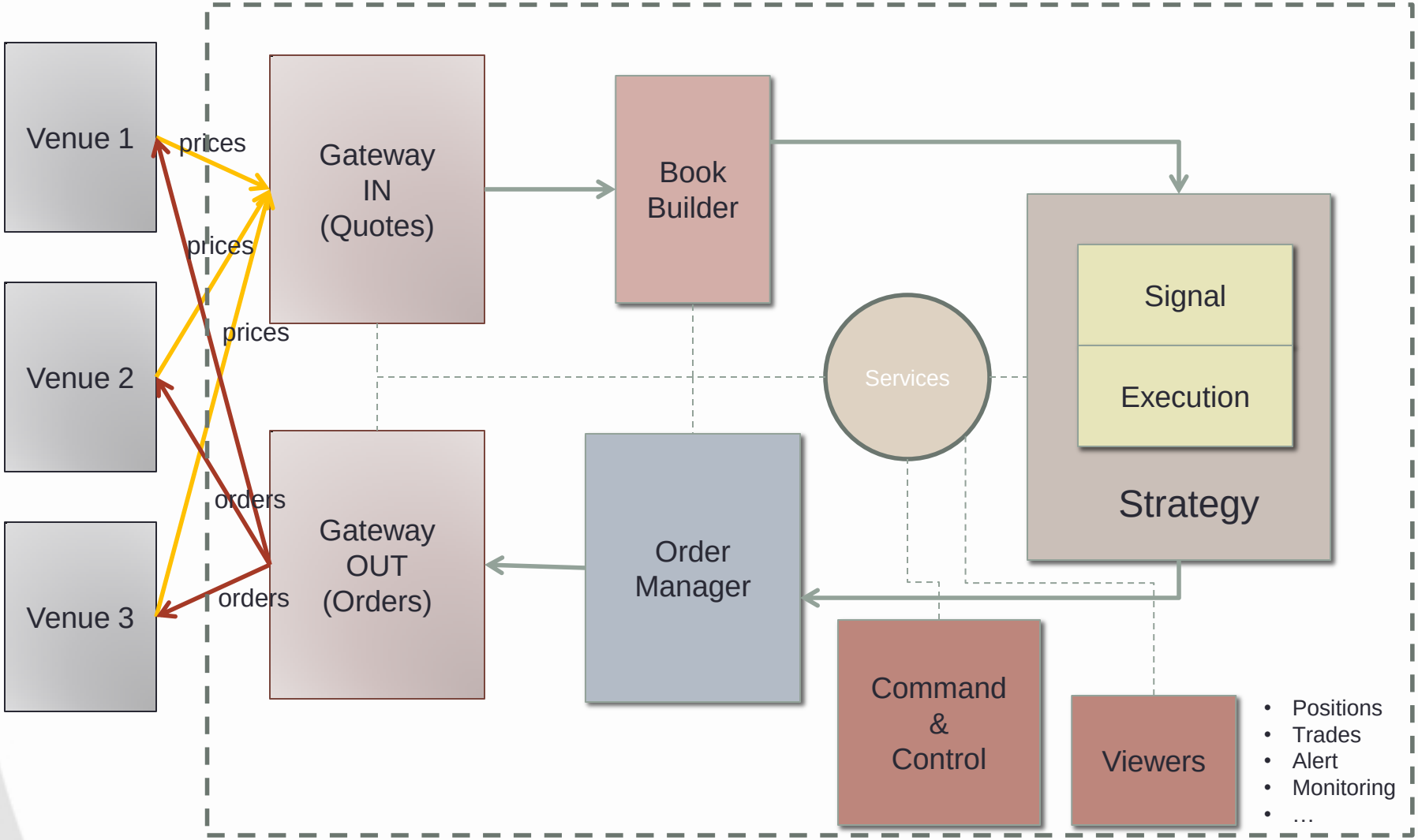
Design – Order Manager

- Position control
- Order control
- Monitor order life-cycle

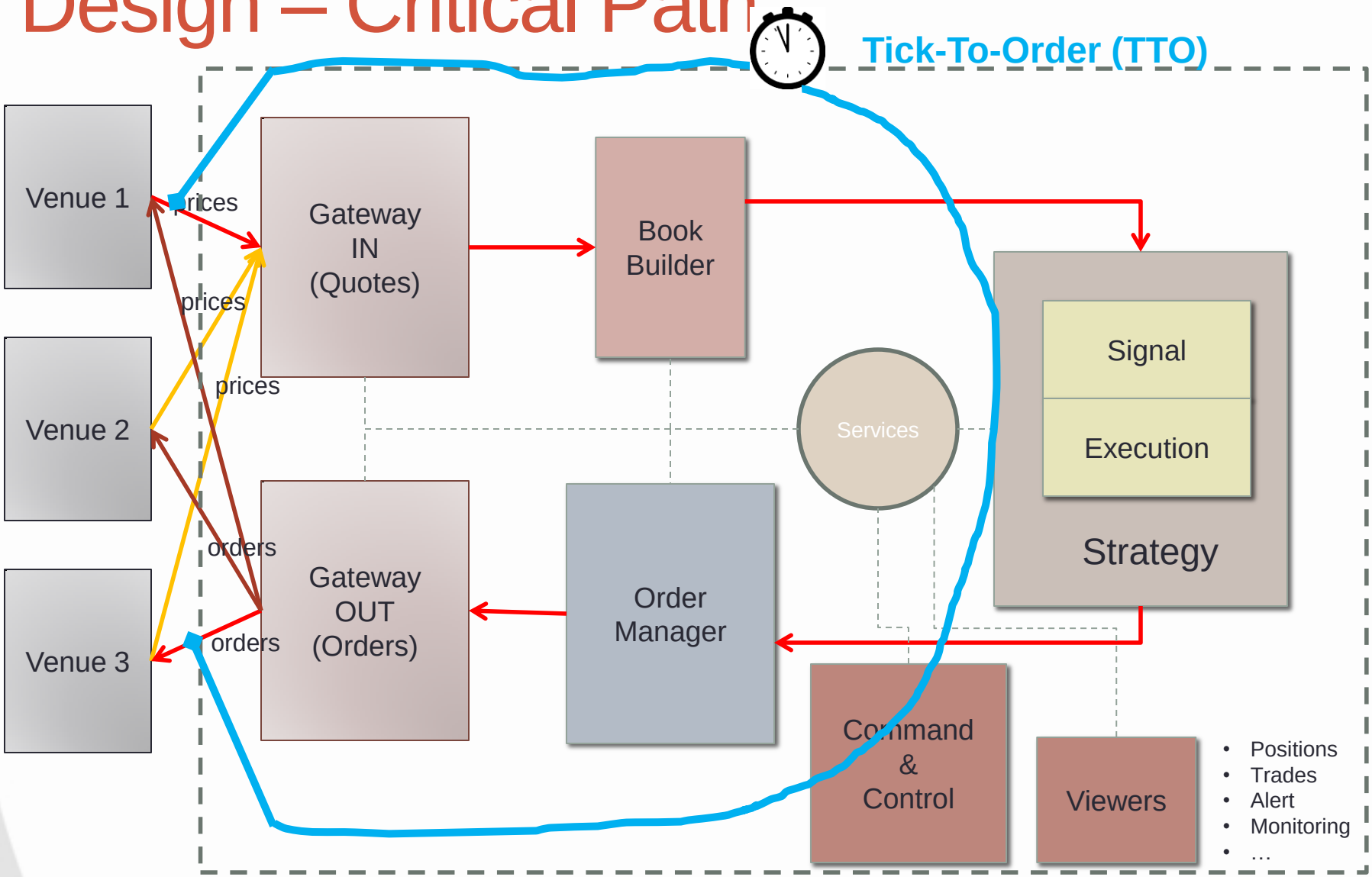
Design – Risk/Compliance manager

- Check exchange rules

Design – Software Overview

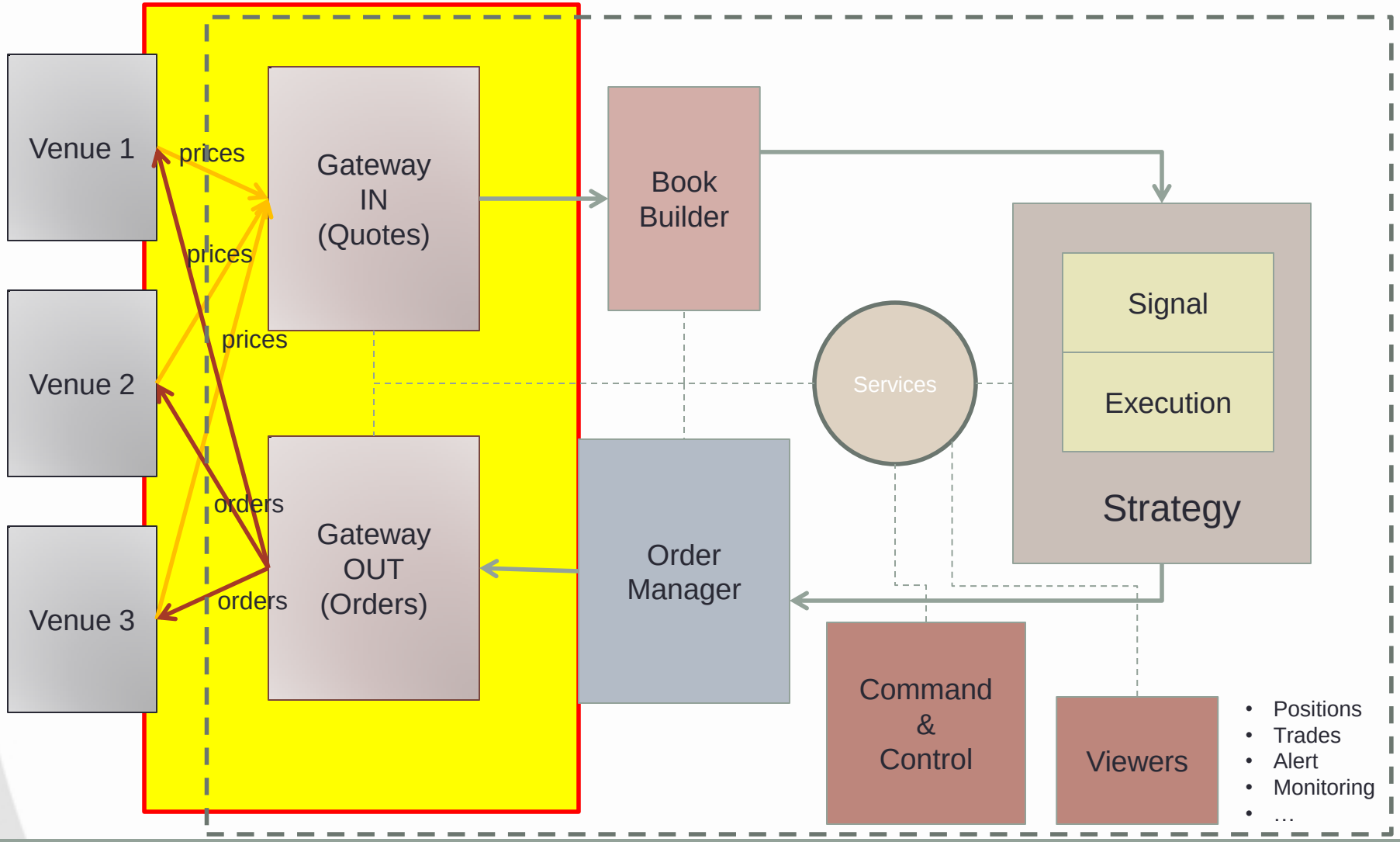


Design – Critical Path



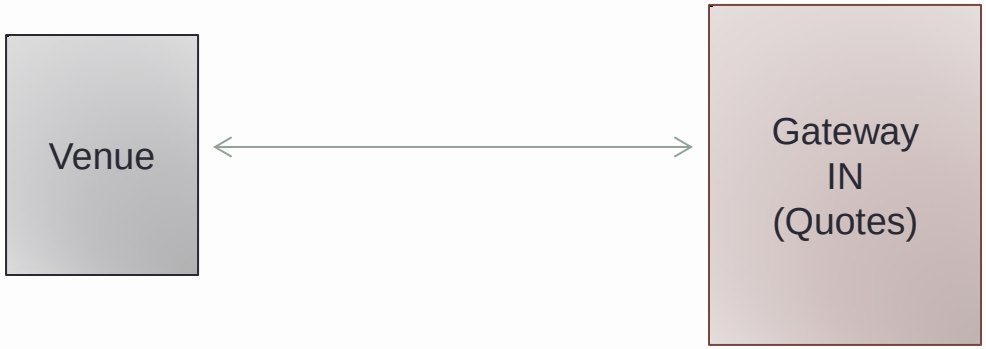
Communication Gateway IN/OUT

Design – Software Overview

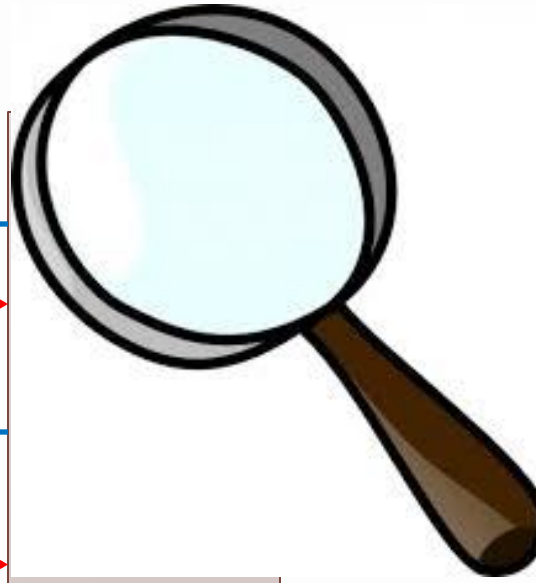


Communication

- Gateway IN:
  - Handle quotes/prices
- Gateway OUT:
  - Handle orders



Protocols - Quote



Hello

Hello back

Please, give the price of MQ, MM, MC?

MQ: S$100 / 5, B$80 /7 MM: S$150 / 10, MC: B$60 / 43

I change my price MQ for $110

MQ: $110 / 5

Someone bought 2 MQ

MQ: $110 / 2

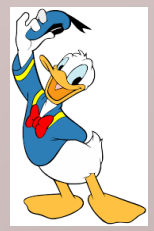
I am running out of MQ

I don't sell MQ any more

Bye

Bye

Gateway IN (Quotes)



Protocols - Quote

Protocols

- We observe an exchange with tags and contents
- Protocols:
  - Communication Initialization (Logon)
  - Communication Acknowledgment (Logon ack)
  - Request Prices for specific symbols (MarketData Request)
  - Send prices (Full Snapshot, Incremental Update)
  - Communication Out (Logout)
  - Communication Out Back (Logout ack)
- Different formats: binary (itch, outch), fix, proprietary protocols

FIX Protocol

- Electronic communication protocol created in 1992
- Designed for real-time exchange
- Created for the equity market, it expanded to Foreign Exchange (FX), Fixed Income (FI), Derivatives (options,…), Clearing
- Many versions (4.2,4.3,4.4,T.1.1,5.0,…)
- FIX Specifications: http://www.fixprotocol.org
- Open protocol
- Platform independent
- 2 types of message: Application/Administrative

Fix protocol format layout

- Tag
  - FIX uses predefined Tags
  - Each Tag represent the specific field
  - Each tag is given a predefined number
  - FIX Field dictionary provides the list of Fields and corresponding Tag numbers (Supplied with Spec)
  - Dictionary is available at the end of specification (by number and by tag name)
- Value
  - Values represent the value of the Tag assigned to
  - Supported Data Types are:

  int, float, char, time, date, data, string

Fix protocol format layout

- All messages start with "8=FIX.x.y
  - Indicates the FIX version of the message being transmitted
  - Useful to support multiple versions
- • All messages terminate with

  "10=nnn<SOH>"
  - nnn represents the Checksum of the data
  - Checksum is the sum of all the binary values in the message
  - Checksum helps to identify the transmission problems

Fix protocol format layout

example:

8=FIX.4.2|9=76|35=A|34=1**|49=DONALD|52=20160617-23:11:55.884|56=VENUE1**|98=0|108=30|141=Y|10=134

- Message fields separated by ASCII 01 (^V A)
- Mandatory header (<FIX4.4): 8 (BeginString), 9(BodyLength), 35(MsgType)
- Mandatory header (>FIX4.4): 8 (BeginString), 9(BodyLength), 35(MsgType), 49(SnderCompID),56(TargetCompID)
- Type is defined by tag 35
- Last Field is the checksum (tag 10)
- Sequence Number (tag 34)

Fix protocol format

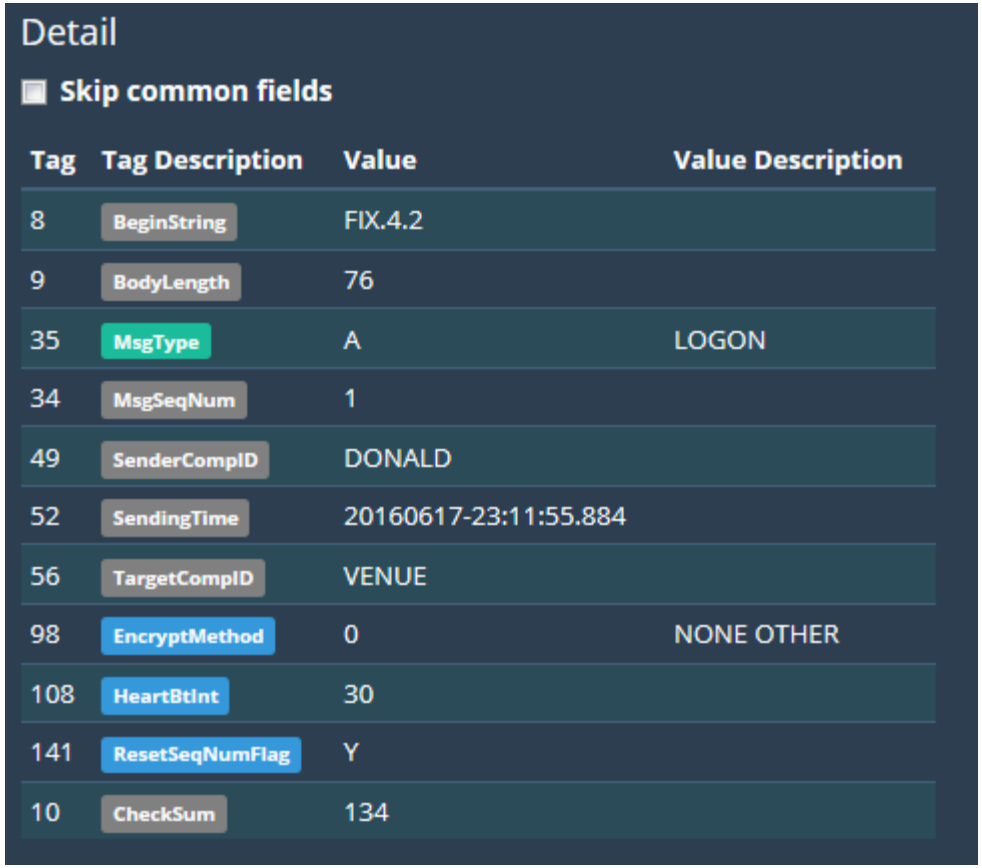
- Body length is the character count starting at tag 35 (included) all the way to tag 10 (excluded).

```
8=FIX.4.2|9=65|35=A|49=SERVER|56=CLIENT|34=177|52=20090107-18:15:16|98=0|108=30|10=062|
     0   + 0  + 5  +   10    +   10    +  7   +        21          + 5  +  7   +   0    = 65
```

- Checksum algorithm of FIX consists of summing up the decimal value of the ASCII representation all the bytes up to but not including the checksum field (which is last) and return the value modulo 256.

FIX messages samples (Logon)

8=FIX.4.2|9=76|35=A|34=1|49=DONALD|52=20160617-23:11:55.884|56=VENUE|98=0|108=30|141=Y|10=134|
https://fixparser.targetcompid.com/



FIX messages samples (MDRequest)

8=FIX.4.2|9=124|35=V|34=2|49=DONALD|52=20160617-23:12:01.333|
56=VENUE|146=3|55=MQ|55=MM|55=MC|262=1|263=1|264=1|265=1|267=2|269=0|269=1|10=088|?
https://fixparser.targetcompid.com/

Tag	Tag Description	Value	Value Description
8	BeginString	FIX.4.2	
9	BodyLength	124	
35	MsgType	V	MARKET DATA REQUEST
34	MsgSeqNum	2	
49	SenderCompID	DONALD	
52	SendingTime	20160617-23:12:01.333	
56	TargetCompID	VENUE	
146	NoRelatedSym	3	
55	Symbol	MQ	
55	Symbol	MM	
55	Symbol	MC	
262	MDReqID	1	
263	SubscriptionRequestType	1	SNAPSHOT PLUS UPDATES
264	MarketDepth	1	
265	MDUpdateType	1	INCREMENTAL REFRESH
267	NoMDEntryTypes	2	
269	MDEntryType	0	BID
269	MDEntryType	1	OFFER
10	CheckSum	088	

FIX messages samples (FullRefresh)

8=FIX.4.2|9=207|35=W|34=2|49=VENUE|52=20160617-23:12:01.336|56=DONALD|55=MQ|268=2|269=0|270=80|271=5|37=1|269=1|270=100|271=7|37=0||37=0|10=196|

https://fixparser.targetcompid.com/

Tag	Tag Description	Value	Value Description
8	BeginString	FIX.4.2	
9	BodyLength	207	
35	MsgType	W	MARKET DATA SNAPSHOT FULL REFRESH
34	MsgSeqNum	2	
49	SenderCompID	VENUE	
52	SendingTime	20160617-23:12:01.336	
56	TargetCompID	DONALD	
55	Symbol	MQ	
268	NoMDEntries	2	
269	MDEntryType	0	BID
270	MDEntryPx	80	
271	MDEntrySize	5	
37	OrderID	1	
269	MDEntryType	1	OFFER
270	MDEntryPx	100	
271	MDEntrySize	7	
37	OrderID	0	
37	OrderID	0	
10	CheckSum	196	

FIX messages samples (Incremental)

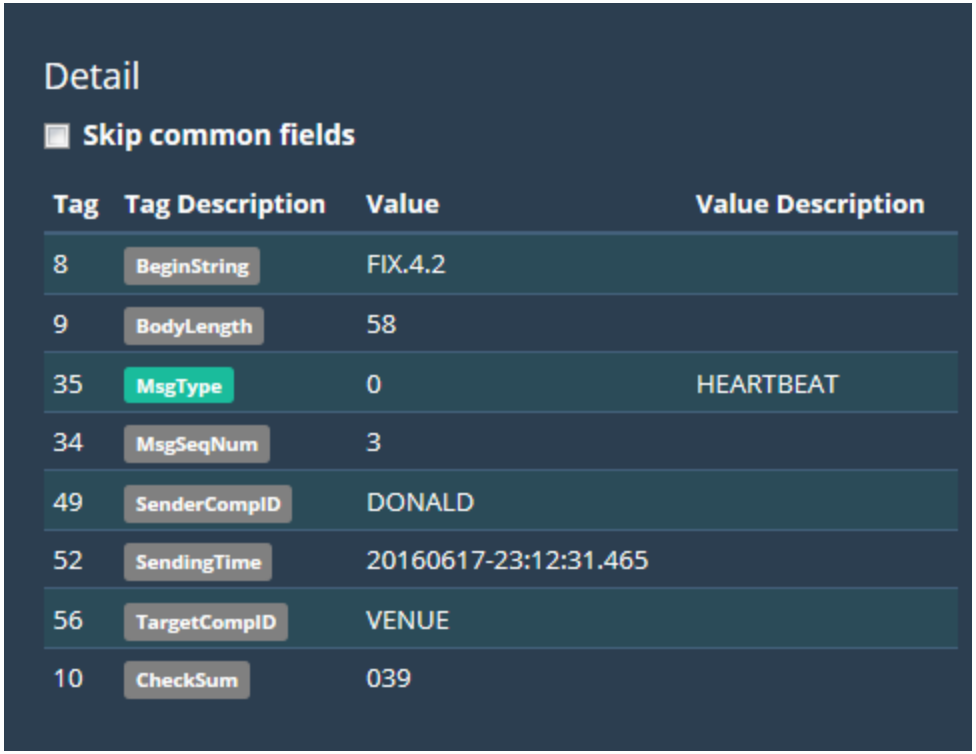
8=FIX.4.2|9=95|35=X|34=5|49=VENUE|52=20160617-23:12:05.551|56=DONALD|268=1|279=1|269=1|270=110|271=5|37=9|10=209|

https://fixparser.targetcompid.com/

Tag	Tag Description	Value	Value Description
8	BeginString	FIX.4.2	
9	BodyLength	95	
35	MsgType	X	MARKET DATA INCREMENTAL REFRESH
34	MsgSeqNum	5	
49	SenderCompID	VENUE	
52	SendingTime	20160617-23:12:05.551	
56	TargetCompID	DONALD	
268	NoMDEntries	1	
279	MDUpdateAction	1	CHANGE
269	MDEntryType	1	OFFER
270	MDEntryPx	110	
271	MDEntrySize	5	
37	OrderID	9	
10	CheckSum	209	

FIX messages samples (Heartbeat)

8=FIX.4.2|9=58|35=0|34=3|49=DONALD|52=20160617-23:12:31.465|56=VENUE|10=039|
https://fixparser.targetcompid.com/



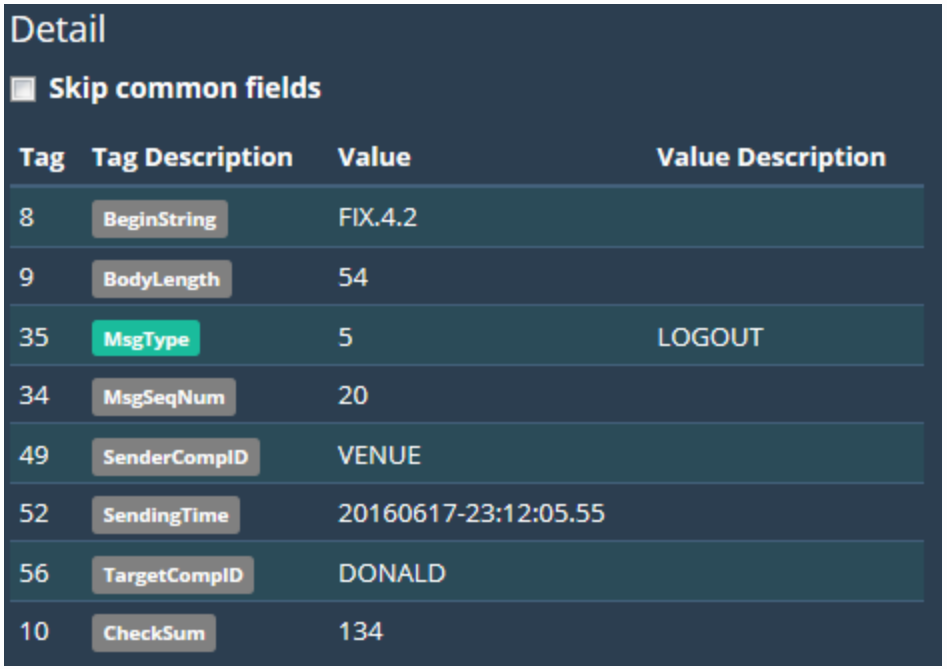
Detail

Skip common fields

Tag	Tag Description	Value	Value Description
8	BeginString	FIX.4.2	
9	BodyLength	58	
35	MsgType	0	HEARTBEAT
34	MsgSeqNum	3	
49	SenderCompID	DONALD	
52	SendingTime	20160617-23:12:31.465	
56	TargetCompID	VENUE	
10	CheckSum	039	

FIX messages samples (Logout)

8=FIX.4.2|9=54|35=5|34=20|49=VENUE|52=20160617-23:12:05.55|56=DONALD|10=134|
https://fixparser.targetcompid.com/



Protocols - Order

Venue

Hello

Hello back

I want to buy 3 MQ for $100

I received your order 3MQ for $100

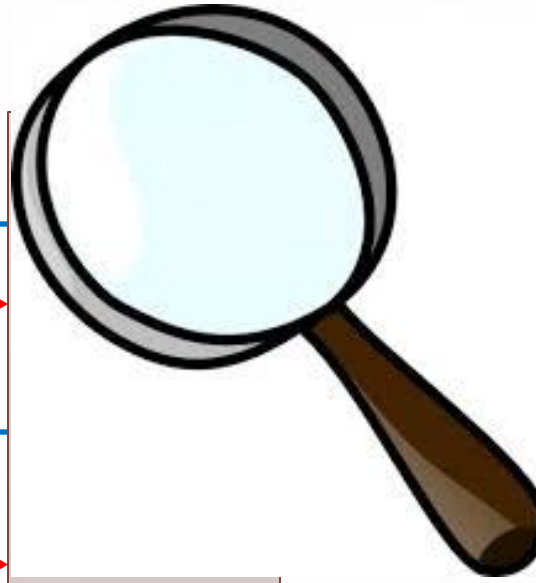
I am selling you 3MQ for $100

I want to buy 1 MQ at market price or cancel

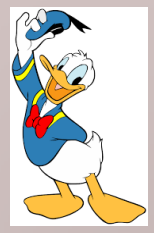
I don't have

Bye

Bye



Gateway
IN
(Quotes)



Protocols - Order

FIX messages samples (NewOrder)

8=FIX.4.2|9=124|35=D|34=2|49=DONALD|52=20160613-22:52:37.227|56=VENUE|11=1|21=3|38=3|40=2|44=100|54=1|55=MQ|60=20160613-22:52:37.227|10=163|

https://fixparser.targetcompid.com/

Tag	Tag Description	Value	Value Description
8	BeginString	FIX.4.2	
9	BodyLength	124	
35	MsgType	D	ORDER SINGLE
34	MsgSeqNum	2	
49	SenderCompID	DONALD	
52	SendingTime	20160613-22:52:37.227	
56	TargetCompID	VENUE	
11	ClOrdID	1	
21	HandlInst	3	MANUAL ORDER
38	OrderQty	3	
40	OrdType	2	LIMIT
44	Price	100	
54	Side	1	BUY
55	Symbol	MQ	
60	TransactTime	20160613-22:52:37.227	
10	CheckSum	163	

FIX messages samples (Execution)

8=FIX.4.2|9=140|35=8|34=2|49=VENUE|
52=20160613-22:52:37.228|56=DONALD|6=10|
11=1|14=100|17=1|20=0|31=10|32=100|37=1|
38=3|39=0|54=1|55=MQ|150=0|151=0|10=184|
https://fixparser.targetcompid.com/

Tag	Tag Description	Value	Value Description
8	BeginString	FIX.4.2	
9	BodyLength	140	
35	MsgType	8	EXECUTION REPORT
34	MsgSeqNum	2	
49	SenderCompID	VENUE	
52	SendingTime	20160613-22:52:37.228	
56	TargetCompID	DONALD	
6	AvgPx	10	
11	ClOrdID	1	
14	CumQty	100	
17	ExecID	1	
20		0	
31	LastPx	10	
32	LastQty	100	
37	OrderID	1	
38	OrderQty	3	
39	OrdStatus	0	NEW
54	Side	1	BUY
55	Symbol	MQ	
150	ExecType	0	NEW
151	LeavesQty	0	
10	CheckSum	184	

Application Message for Orders

- Trade Messages
  - New Order (Single)
  - Execution Report
  - Order Cancel Request
  - Order Cancel/Replace Request
  - Order Status Request etc

Administrative Messages

- Logon - Starts the Session
- Heartbeat – Used to check the health in case of idle
- Test Request
- Resend Req
- Logout

Communication Model

- Session based communication
  - Session is communication between two parties
- Initiator / Client
  - – party who initiates the communication
- Acceptor /Server
  - – party who receives connection request from Initiator
  - – Server validates client request using login message

FIX Session

- FIX is a session protocol
  - Each session maintains the bi-directional messages between two parties
  - Session can spread across multiple physical connections
  - Session is maintained using sequence number
  - Both parties rely on sequence numbers to maintain the orderly communication
  - Every new session starts with sequence number 1
  - Missing messages are re-transmitted with bi-lateral agreement between both parties

FIX parser implementation

- FIX Engine:
  - Create your own parser (regular coding, FPGA,…)
  - Use a library (quickfix)
- Caution: Parsing is on the critical path, therefore speed matters
- Architecture: Client/Server

Naive parsers

```
Python example (132300 messages)

def parse_message(message):
   order_parsed=dict((int(k),v) for k, v in (e.split('=')
       for e in   message.split('\01') if len(e)>0))


 infile = open(args[0],'rb')
 for line in infile:
   if "8=FIX" in line:
      parse_message(line.rstrip('\n').split(',')[0])

$ python time python test.py /tmp/test.log > /tmp/test.fix
real    0m2.290s
user    0m2.241s
sys     0m0.044s

C++ Example:
$ test.bin /tmp/test.log > /tmp/test.fix
real    0m0.994s
user    0m0.956s
sys     0m0.056s
```

16 microseconds / message

6 microseconds / message

Commercial FIX Engines

- NYFIX – Appea
- Aegisoft – Aethna
- Reuters – Traid
- Financial Fusion (Sybase) - TradeForce
- CameronFIX (Orc Software)

- QuickFix – Open Source FIX Engine (C++/Java)

Quickfix implementation

- Library to download http://www.quickfixengine.org/
- C++ / Java / Python

```
import sys
import time
import quickfix as fix
import quickfix42 as fix42

class Application(fix.Application):
    def onCreate(self, sessionID): return
    def onLogon(self, sessionID):
            self.sessionID = sessionID
            print ("Successful Logon to session '%s'." % sessionID.toString())
            return
    def onLogout(self, sessionID): return
    def toAdmin(self, sessionID, message):return
    def fromAdmin(self, sessionID, message):return
    def toApp(self, sessionID, message):
        print "Sent the following message: %s" % message.toString()
        return
    def fromApp(self, message, sessionID):
        print "Received the following message: %s" % message.toString()
        return
```

Quickfix implementation

```
def main(config_file):
    try:
        settings = fix.SessionSettings(config_file)
        application = Application()
        storeFactory = fix.FileStoreFactory(settings)
        logFactory = fix.FileLogFactory(settings)
        initiator = fix.SocketInitiator(application, storeFactory, settings,
logFactory)
        initiator.start()

        while 1:
            st_input = raw_input()
            # command

if __name__=='__main__':
    parser = argparse.ArgumentParser(description='FIX Client')
    parser.add_argument('file_name', type=str, help='Name of configuration file')
    args = parser.parse_args()
    main(args.file_name)
```

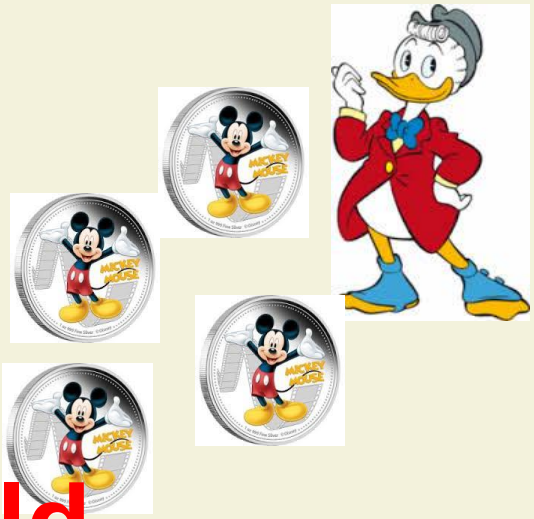
Book building

Motivation (Use a less risky solution)

At a given time, Donald already **knows** that he can buy for $100 and sells for $110

Wants to sell 4 MQ for $110

seller



Wants to sell MQ for $100

New York

Gladstone

Chicago

buyer

Wants to buy MQ for $110

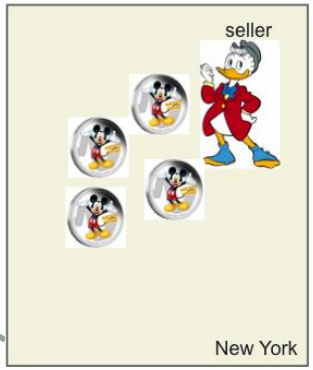
Gus

How does Donald know this information?

Motivation: Building a book

At a given time,
Donald already **knows**
that he can buy for $100
and sells for $110

BID				OFFER			
ID	Time	Volume	Price	Price	Volume	Time	ID
2	7/4/16 11:08	1	110	110	4	7/4/16 11:06	1



Wants to sell 4 MQ
for $110

Motivation: Bookbuilding

buyer
seller

Mickey

seller

London

seller

New York

Gladstone

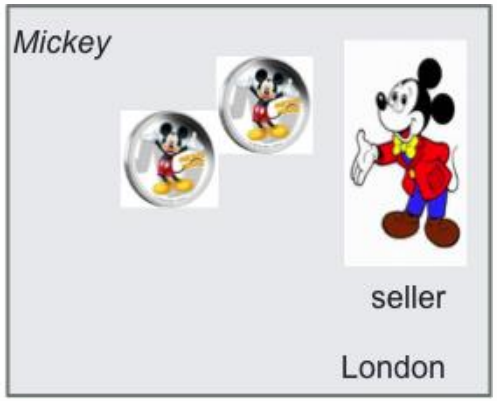
buyer

Chicago

Wants to buy MQ
for $110

Gus

Motivation: Bookbuilding



buyer
seller

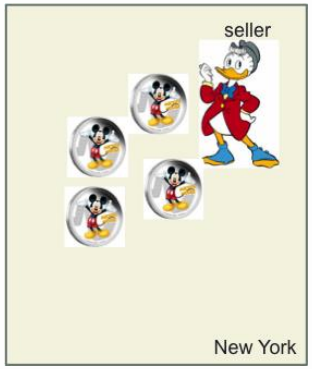
Mickey
Wants to sell 2 MQ
for $90

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	GUS	7/4/16 11:25	1	110	110	4	7/4/16 11:25	GOLD	1
					90	2	7/4/16 11:25	MICK	3

Donald will buy for $110?

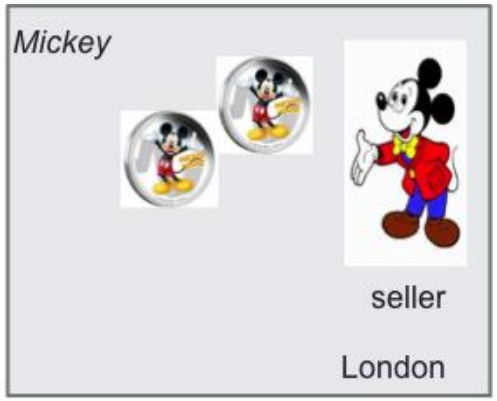
Gus

Wants to sell 4 MQ
for $110



Gladstone

Motivation: Bookbuilding



buyer
seller

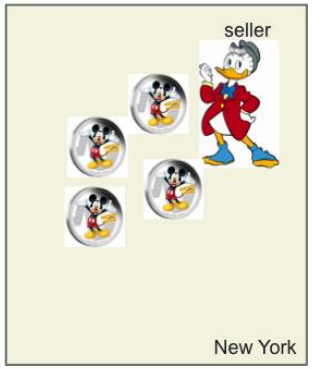
Mickey
Wants to sell 2 MQ
for $90

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	GUS	7/4/16 11:25	1	110	90	2	7/4/16 11:25	MICK	3
					110	4	7/4/16 11:25	GOLD	1

No, he will sort the prices first!

Gus

Wants to sell 4 MQ
for $110

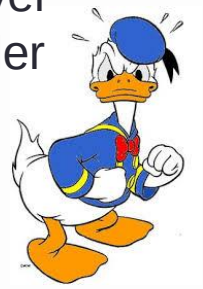


Gladstone

Motivation: Need a fast book

Mickey

buyer
seller



BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	GUS	7/4/16 11:25	1	110	90	2	7/4/16 11:25	MICK	3
					110	4	7/4/16 11:25	GOLD	1

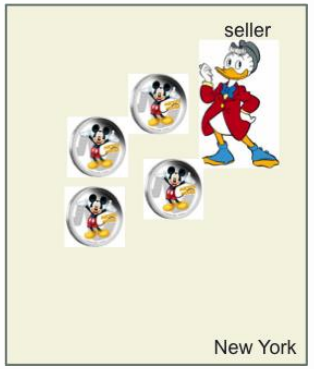
faster

faster

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	GUS	7/4/16 11:25	1	110	90	2	7/4/16 11:25	MICK	3
					110	4	7/4/16 11:25	GOLD	1

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	GUS	7/4/16 11:25	1	110	90	2	7/4/16 11:25	MICK	3
					110	4	7/4/16 11:25	GOLD	1

Gus



Gladstone

Motivation: Need optimized bookbuilding with more assets and venues


MQ $100

SYMBOL: MQ

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	GUS	7/4/16 11:43	1	110	90	4	7/4/16 11:43	GUS	1
					110	2	7/4/16 11:43	GOLD	3


MM $150

MM $160

SYMBOL: MM

BID					OFFER				
ID	Venue	Time	Volume	Price	Price	Volume	Time	Venue	ID
2	PIC	7/4/16 11:40	1	110	90	2	7/4/16 11:40	DAIS	3
4	MIN	7/4/16 11:40	4	105	110	4	7/4/16 11:40	MICKEY	1
5	PIC	7/4/16 11:40	10	95					


MQ $80

buyer

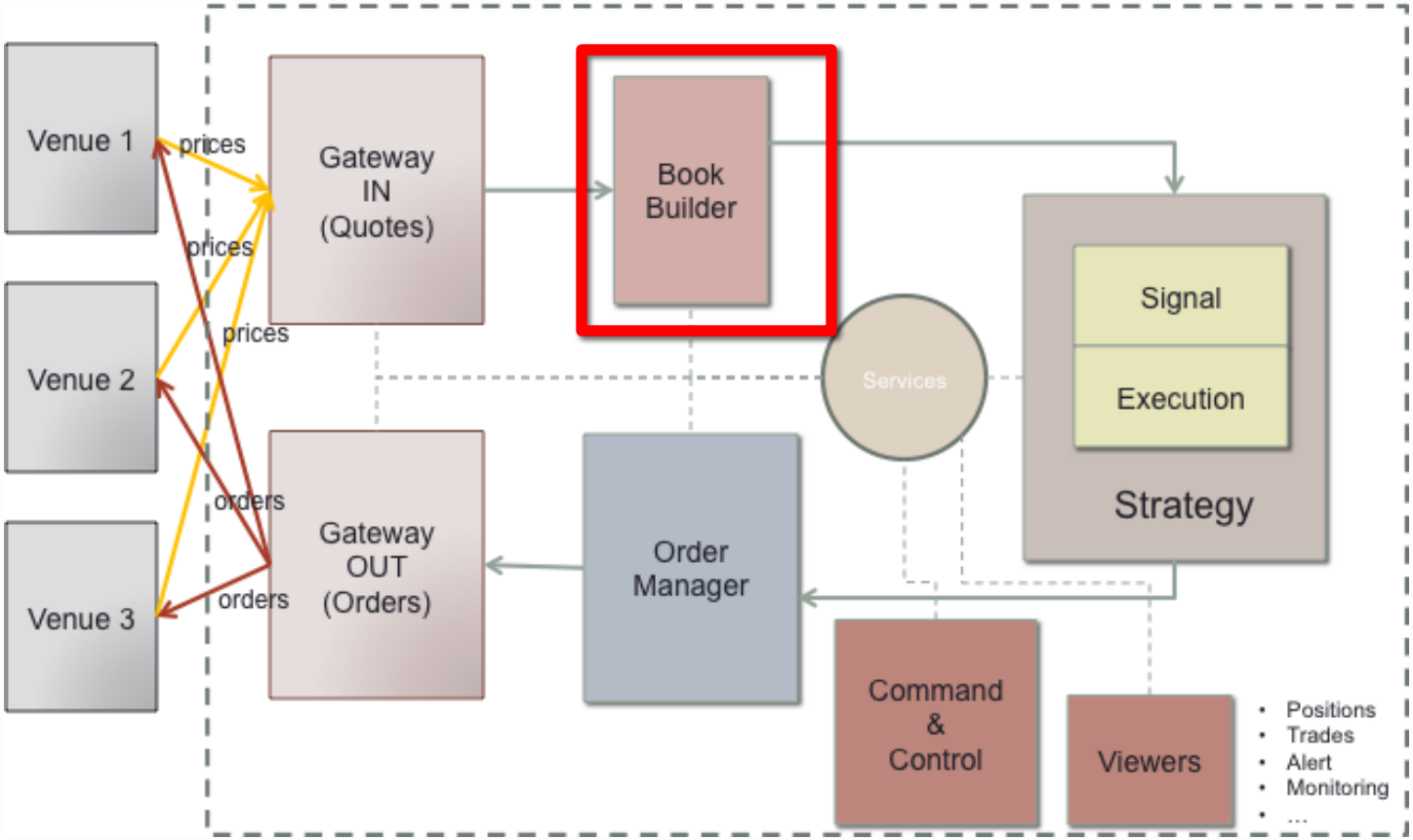
buyer
seller


MM $130

Motivation: Volume

- A gigantic volume a day: 20+ GB / day
- 3 MB/s (or even more)
- 20 bytes (message), handling 200,000 message / second
- During the day, the number of messages received varies a lot

Book Builder



Bookbuilder design

- Critical component: Primary source of market information for trading models
- 3 main operations:
  - Add
  - Cancel
  - Modify (cancel and add)
- Goal: Implement these operation in O(1) time
- Most used functions?
  - What are the best bid and offer?
  - How much volume is there between prices A and B?
  - How many levels we have for this symbol?
- Indexed by price, order id

Bookbuilder MarketData management

- Add quote
- Modify quote
- Delete quote

Which data structure?

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	-	O(1)	O(1)	O(1)	-	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	-	O(log(n))	O(log(n))	O(log(n))	-	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)

Idea

- Binary tree by price, each of which is itself a doubly linked list of Order objects.
- Each side of the book, the Bid and the Offer should be in separate trees so that the inside of the book corresponds to the end and beginning of the Bid Offer tree, respectively.
- Each quote/order is also an entry in a map keyed off id number

Data structure

```
Quote
   int idNumber;
   bool buyOrSell;
   int shares;
   int limit;
   int entryTime;
   int eventTime;
   Quote *nextOrder;
   Quote *prevOrder;
   int venue;
```