恒生极速交易系统相关的技术架构与应用发展

朱金奇

恒生电子股份有限公司　研发中心　高级技术专家

自我介绍

朱金奇

恒生电子股份有限公司 研发中心

2011年加入恒生电子

目前负责恒生高性能中间件平台的研发与推广，平台专注于低延时、高可用等业务场景

目录

1.为什么需要极速交易

2.恒生极速交易系统发展的重要时间节点

3.恒生极速交易系统设计遇到哪些问题？如何解决

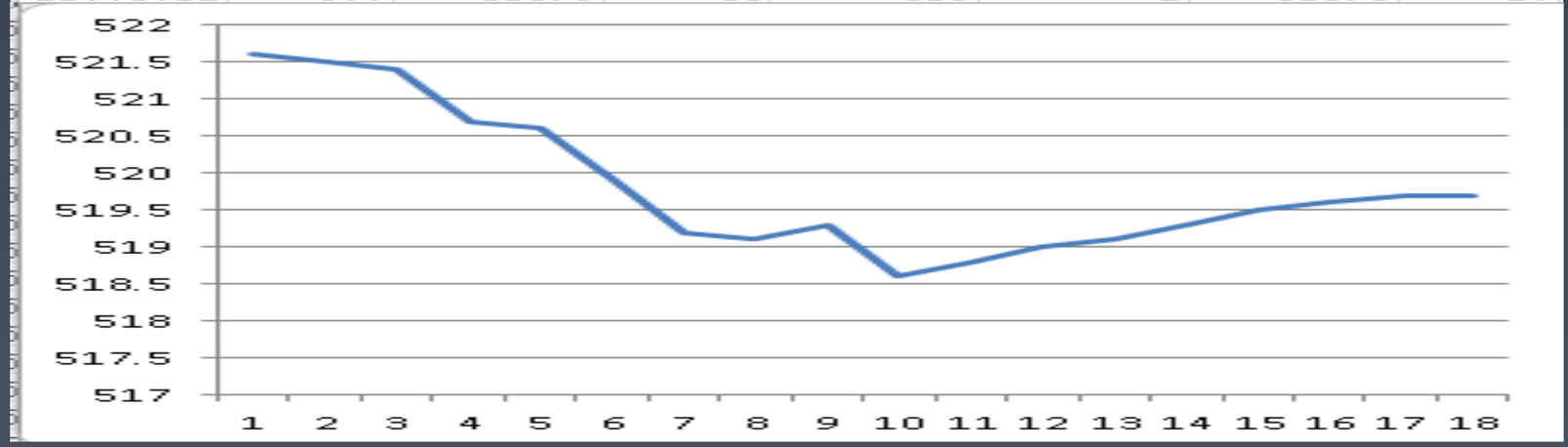
4.怎样达到纳秒级的速度

为什么需要极速交易？

- 交易所撮合原则：价格优先、时间优先
- 高频策略交易在市场的交易份额逐年提升

交易场景：Tick行情报价

时间	毫秒	成交价格	成交量	卖一价	卖一量	买一价	买一量
21:03:06	0	522.2	4	522.3	9	522.2	19
21:03:06	500	522.3	32	522.3	11	522.2	3
21:03:07	0	521.5	164	521.6	524	521.5	6
21:03:07	500	521.4	320	521.5	34	521.2	1
21:03:08	0	521.4	72	521.4	14	521.3	11
21:03:08	500	520.7	246	520.7	416	520.6	8
21:03:09	0	520.6	138	520.6	7	520.5	1
21:03:09	500	519.9	342	519.9	33	519.8	2
21:03:10	0	519.2	322	519.2	126	519.7	3
21:03:10	500	519.1	390	519.1	23	519	11
21:03:11	0	519.3	256	519.3	6	519.2	12
21:03:11	500	518.6	254	518.6	25	518.5	
21:03:12	0	518.7	240	518.8	6	518.7	3
21:03:12	500	518.9	44	519	2	518.8	20



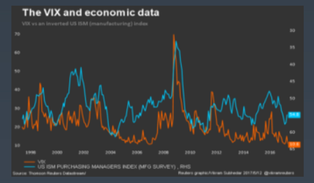
突发大量卖单

速度快的交易者可以立即521.5卖出开仓并成交

速度慢的交易者521.5卖出开仓，挂在卖一价，后续没有成交机会，错失交易时机

交易场景关注哪些时间？

- 客户端响应时间
- 订单上行时间
- 成交/行情 下行时间



策略终端

金融机构
订单系统

交易所
撮合系统

我们是怎么去做的？

- 客户分类
- 业务分类
- 系统分类

专业投资者

极速业务

行情、成交主推

普通投资者

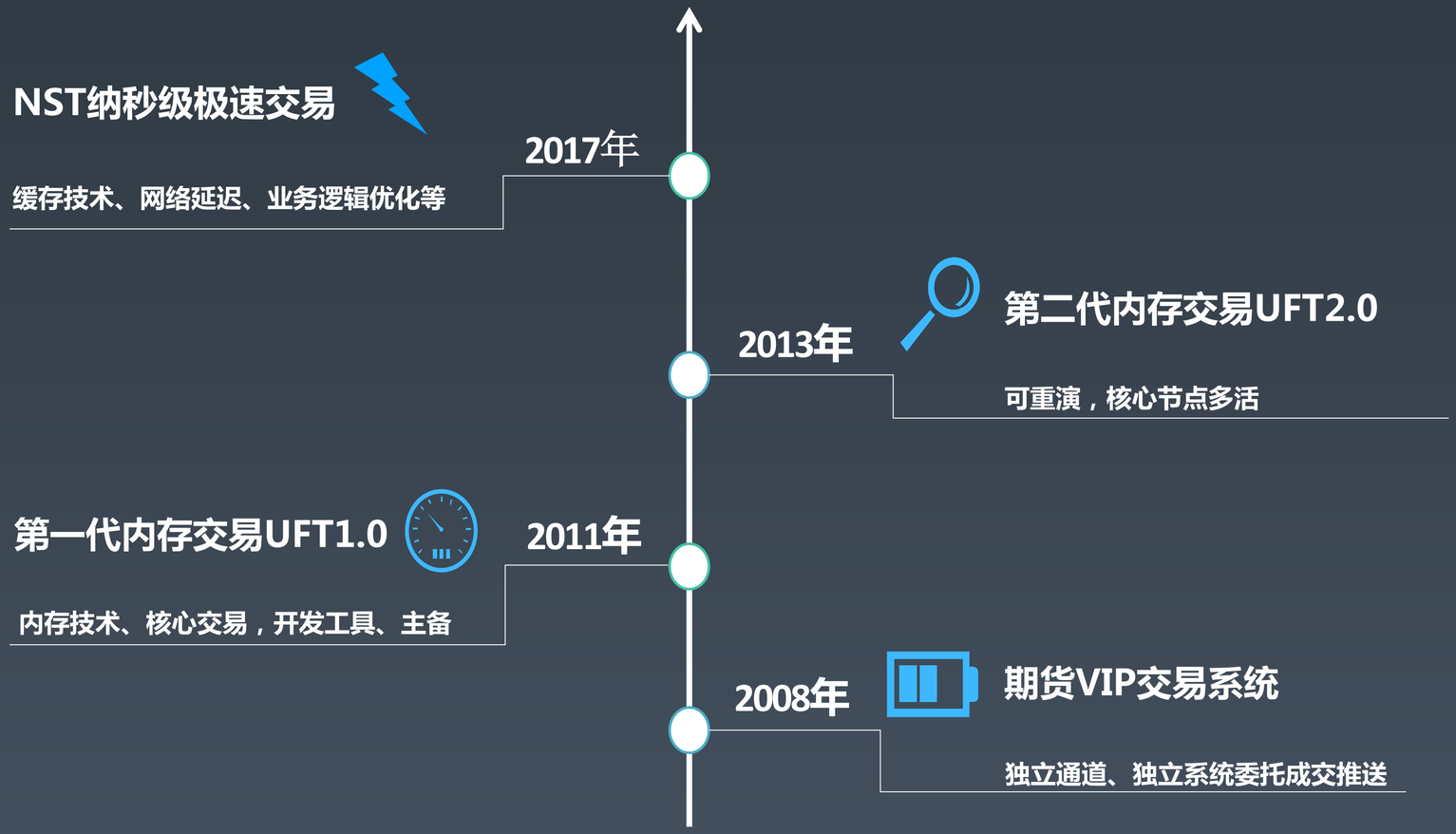
普通业务

行情、成交轮询

内存交易系统

数据库交易系统

恒生极速交易：从毫秒、到微秒、再到纳秒的飞跃



第一代UFT（Ultra Fast Trading）总体架构

订单服务处理耗时从

20ms降低到**300us**以内

客户端响应时间

<5ms

第一代UFT遇到了哪些问题

- **开发**：新业务开发周期长，开发难度高？
- **性能**：客户数量多，交易量增大，查找性能变慢？
  客户风控风控条目增多，延时就变大？
- **管理**：数据全在内存中，管理不方便？
- **运维**：问题排查很麻烦？

怎么让开发更加快速，程序更加稳定

◆专门针对金融行业的内存数据库，支持统一的访问API，业务与数据分离，业务开发人员不需要关心内存数据的实现

◆ 支持专门的开发工具快速开发业务，提高系统稳定性，插件化开发，业务功能可任意组合、扩展

开发工具帮我们做了哪些事情？

内存表结构设计　接口管理　元数据管理　伪代码翻译

提高开发效率

- 原数据管理：标准字段、标准错误号、数据字典、内存对象等
- 接口管理：服务接口、函数接口
- 基础数据管理：系统配置参数

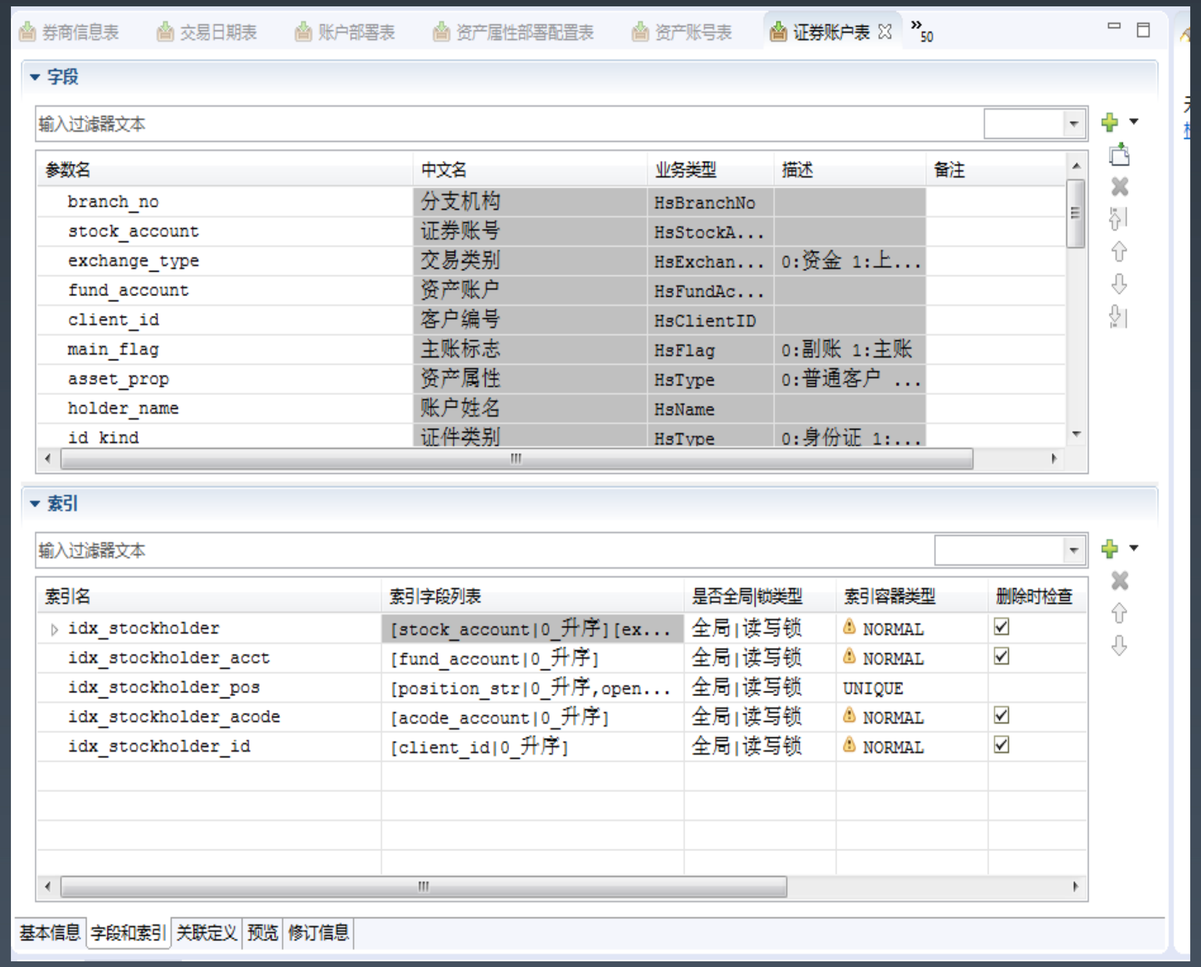
降低开发难度

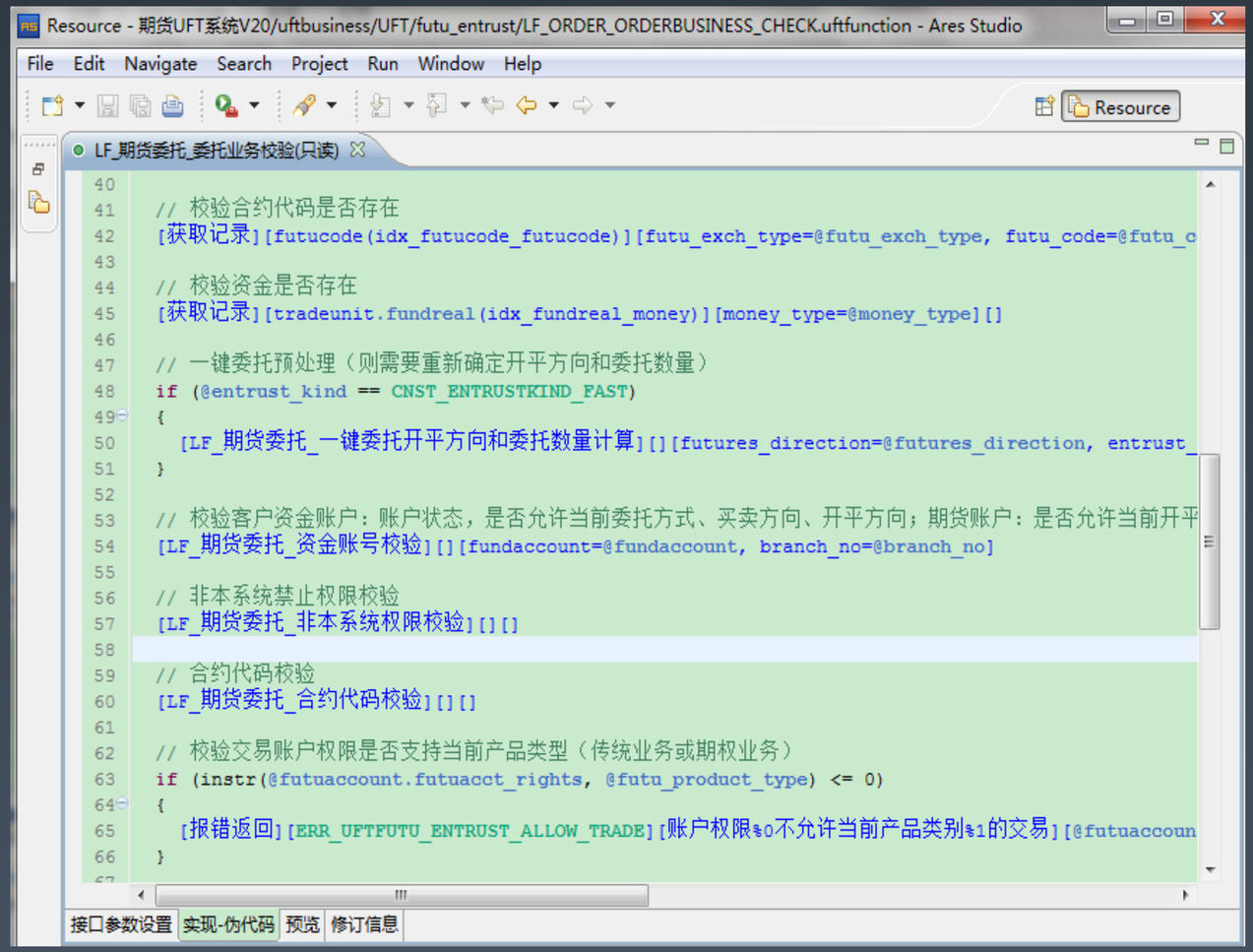
- 业务伪代码代码开发（系统宏：[插入记录]、 [修改记录] 等）
- 禁止关键字：new/malloc、goto等
- 集成pclint等静态代码检查工具
- 死锁分析检查

规范开发过程

- 一键生成代码、上传、编译、运行、伪代码调试
- 自动化测试对接，自动分析修改影响到的业务接口
- 业务逻辑分层（逻辑层、原子层）

UFT开发工具帮我们做了哪些事情？





数据量大了性能还能否保持稳定？

- ✓ 交易单元，数据预先关联（1：1，1：N，N ：1，N ：N）
- ✓ 主体呈树状组织，缩小查找范围

风控条目变多了，延时就变大？

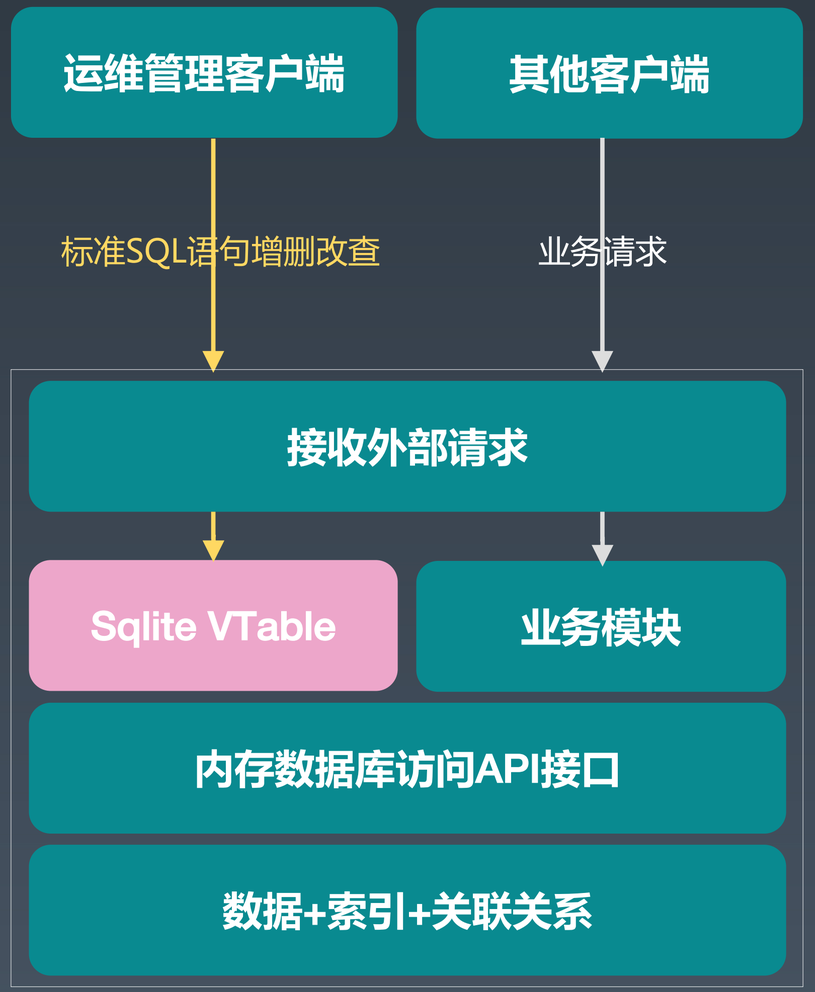
- ✓拆分
- ✓并行
- ✓汇总

数据全在内存，怎么管理？

SQLite Virtual Table

是一种自定义的扩展，允许用户通过代码定制表的数据结构和数据内容

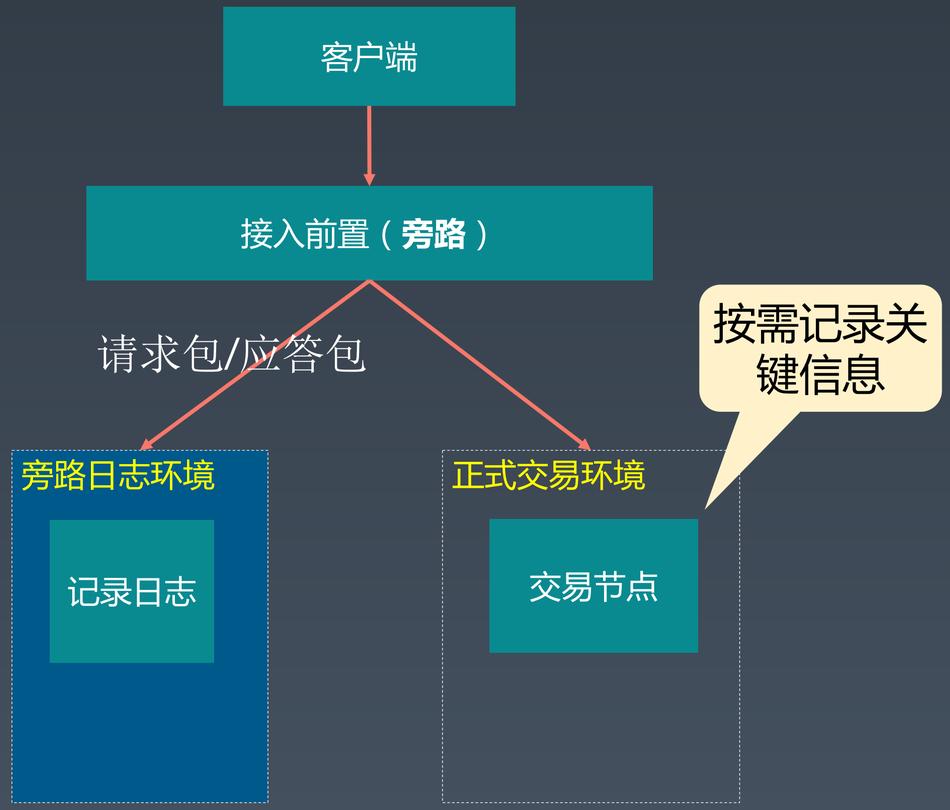
对于数据库引擎，它和普通表一样，允许进行大多数的sql操作



问题怎么排查？

1. 按需记录

2. 旁路记录



第一代UFT解决了这些问题

高性能

封装内存数据库，支持事务、索引，业务内存中处理

易管理

通过标准SQL进行管理，采用灵活的日志方式排查问题

易开发

开发工具开发业务，规范业务代码，提升开发效率和程序稳定性

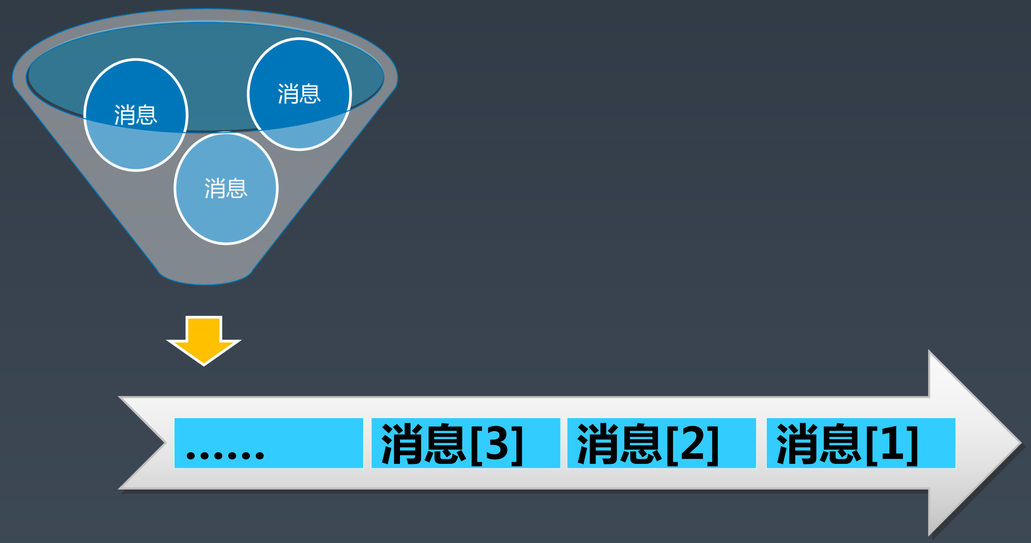
高可用

交易核心节点主备部署，实时同步，支持秒级切换

新三板、港交所
交易所场景需求

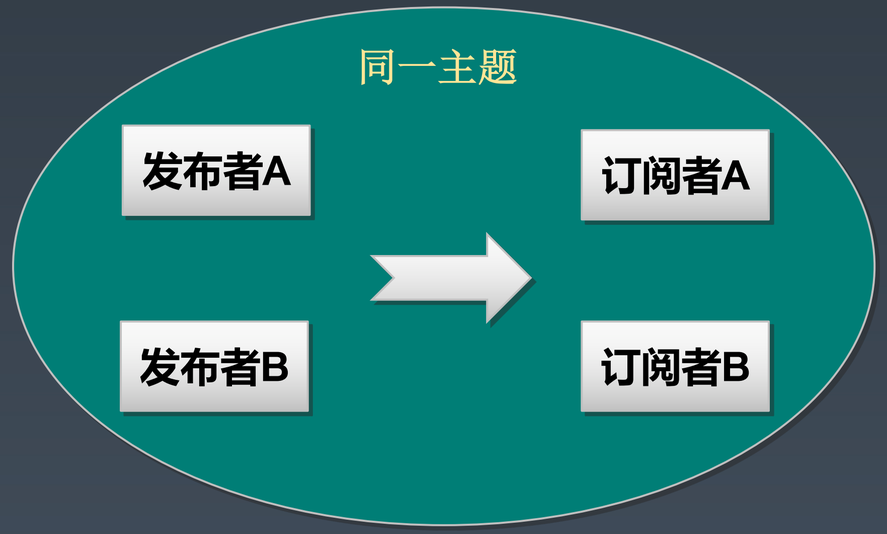
核心节点（撮合）可重放

排队机保证输入的顺序一致



- 将所有业务请求按时序、优先级进行编排，确保各核心数据处理一致
- 可以进行业务反演，用于交易服务器宕机后反演数据恢复，升级时也可进行数据核对

可靠组播保证消息有序可靠传输



- 采用基于组播协议，增加处理同一消息的组件对性能无影响
- 采用可靠、有序的数据发布/订阅模式，简化系统间耦合程度
- 支持一对多和多对多模式，订阅收到同一主题多个发布者发布的数据后过滤
- 采用A/B网，发送者在发送消息的时候，向A网卡发送一次，同时向B网卡发送一次，在硬件上保证通讯可靠

第二代UFT总体架构

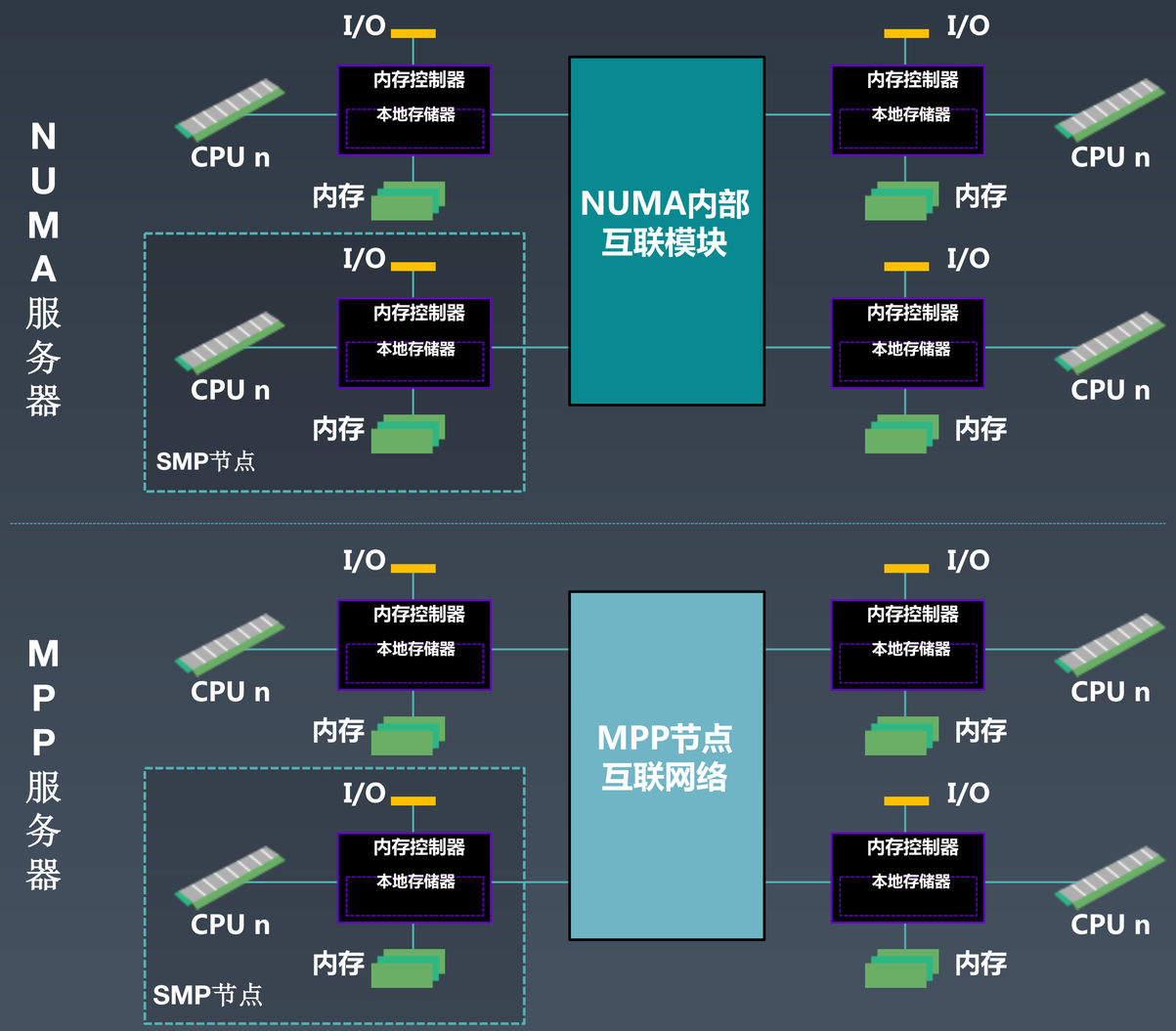
- 主排队机对业务处理请求进行排队
- 依靠可靠组播，多个撮合核心同时订阅并且排队机的消息
- 提供差异化的实现，查询请求无需排队

速度是否可以更快一些？

选择方案：

复杂的交易所API处理和交易核心采用 CPU加速。重复性强的周边策略、行情、风控采用FPGA加速

服务器三种体系架构

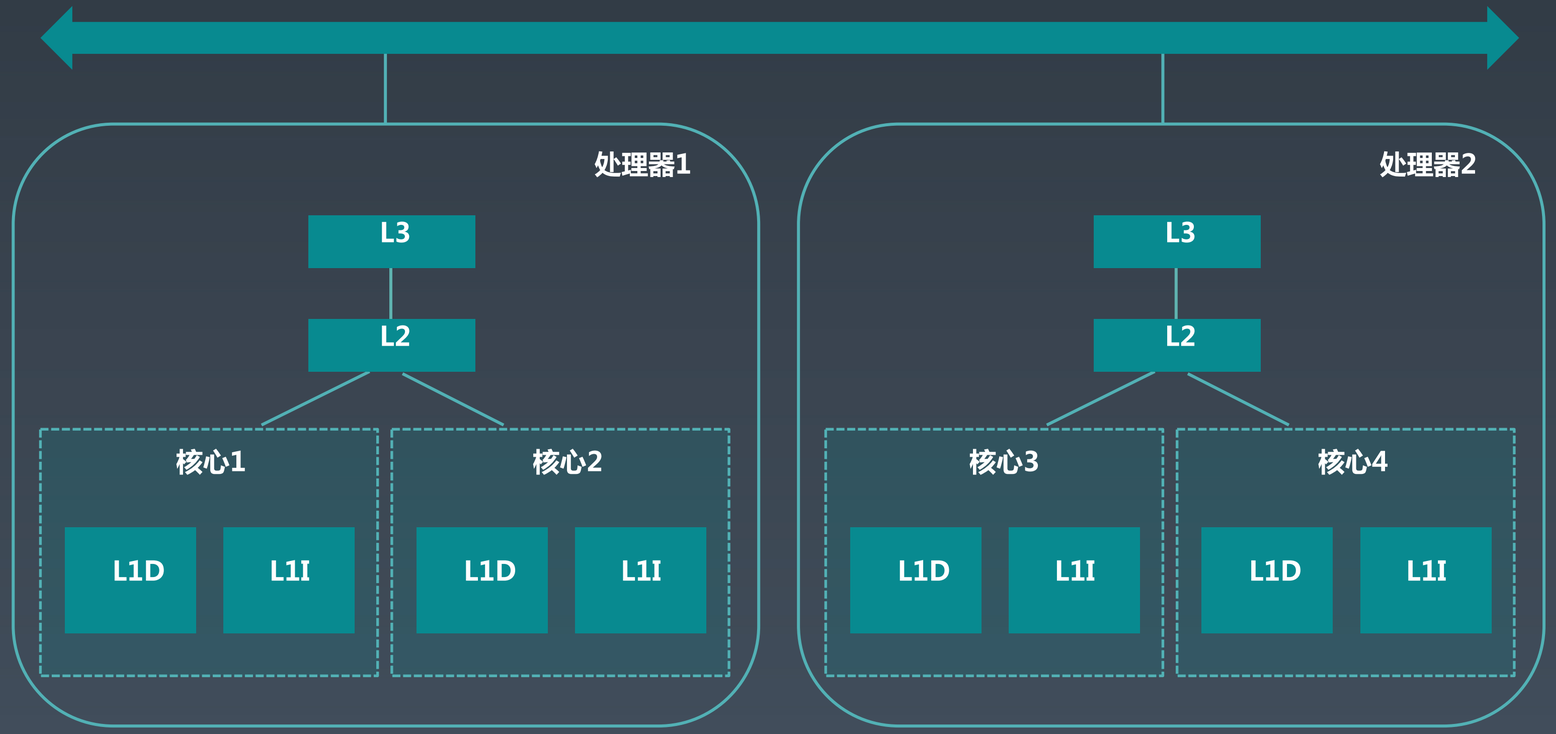


SMP
对称多处理器结构

MPP
海量并行处理

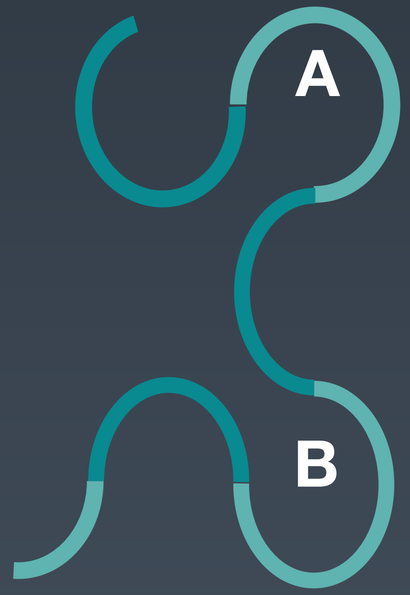
NUMA
非一致存储访问结构

CPU内部缓存结构分析



存储类型	延迟时间 (ns)	存储大小
寄存器	<=1	几字节
L1d	~1	几十K
L2	~10	几百K
L3	~40	几M
主存	~100	几G

降低访存延迟—考虑物理架构



充分考虑主机NUMA架构

充分考虑缓存和CPU核心的关系

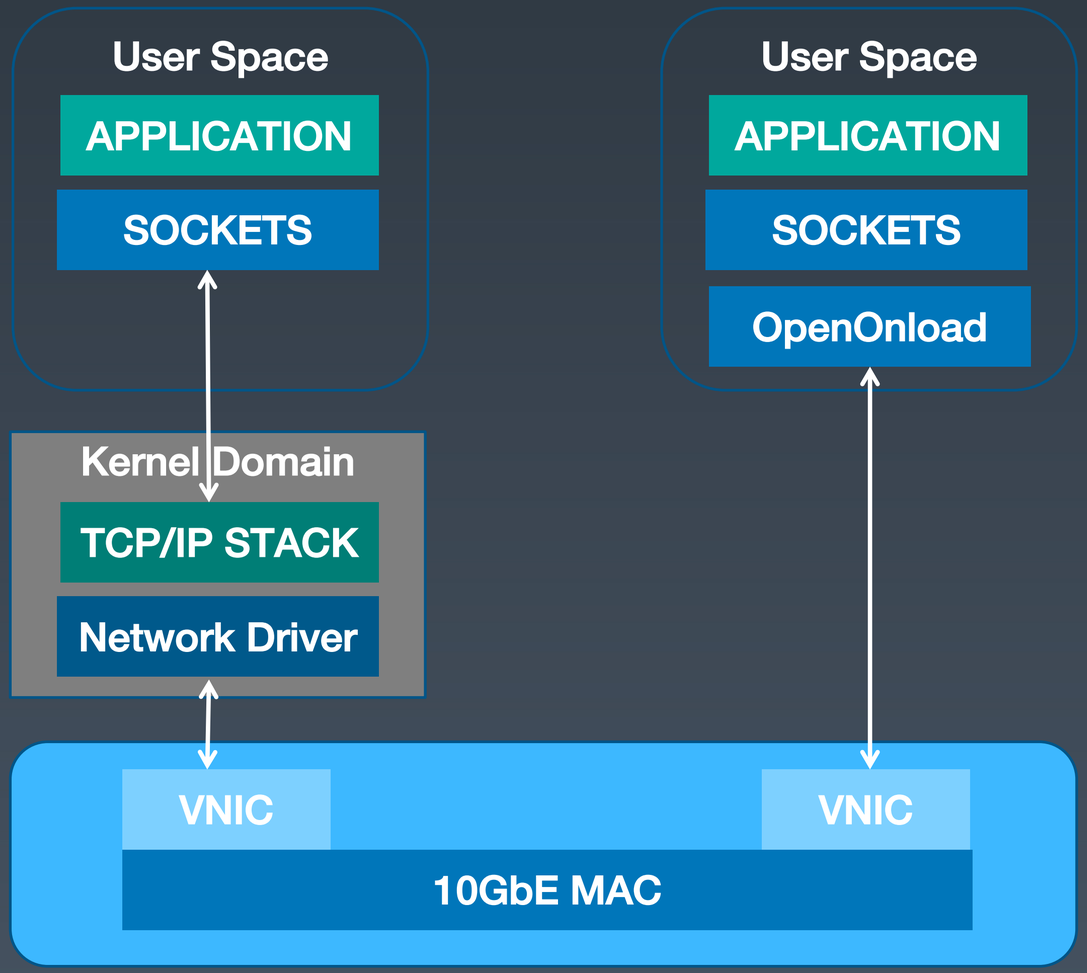
降低访存延迟—提高缓存命中率

- 连续 vs 离散
- 数据结构的定义和业务访问顺序一致
- 某些数据结构按照缓存线对齐原则
- 针对业务特色组织数据结构

什么样的查找算法更快？

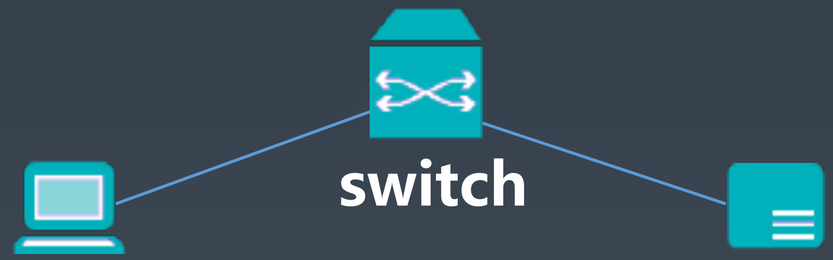
无需查找的“查找算法”就是最快的查找算法！

低延时网卡机制分析



低延时网卡优化

- 内核旁路(Kernel bypass)
- 用硬件替代软件(offload)

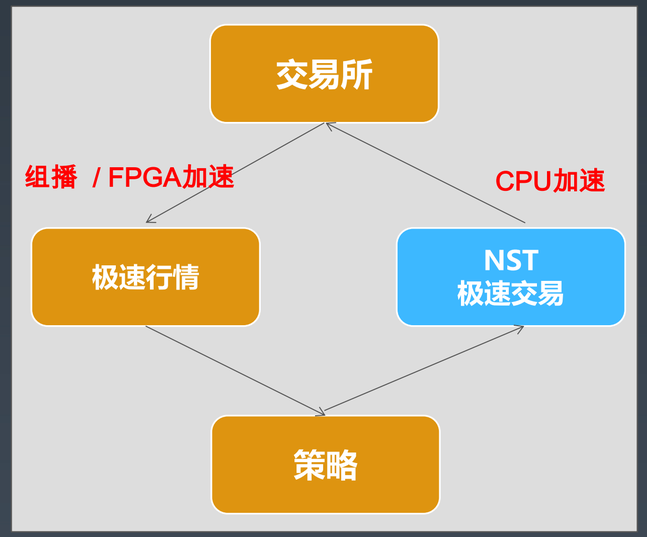


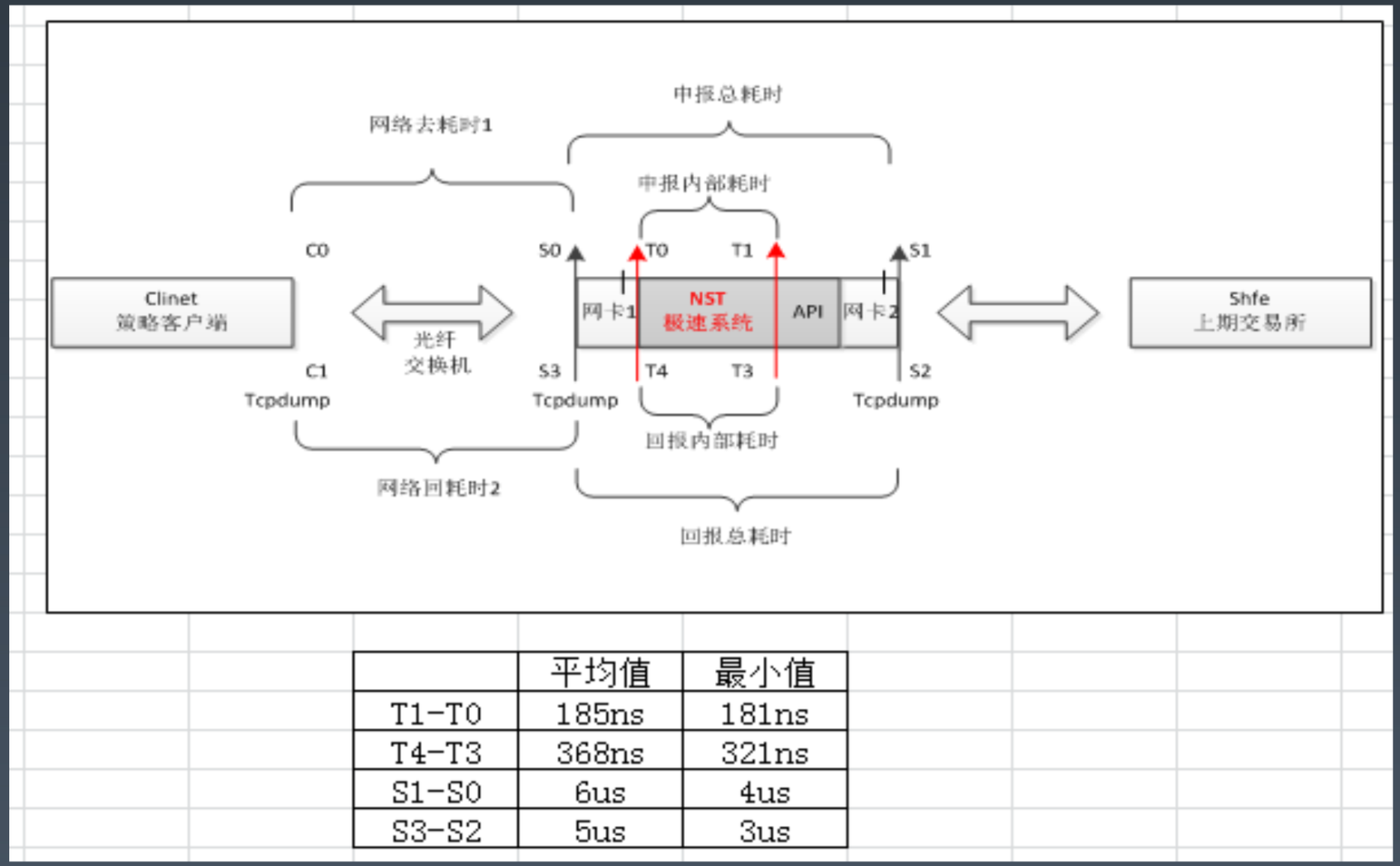
网络延迟时间

1000M网络	40 — 100 us
10G网络	~ 15 us
低延时网卡	2.0 — 4.3 us

恒生第一代纳秒级交易产品：期货NST

交易系统内部时延：NST交易核心处理时延<300纳秒





	平均值	最小值
T1-T0	185ns	181ns
T4-T3	368ns	321ns
S1-S0	6us	4us
S3-S2	5us	3us

Geekbang 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

THANKS!

QCon 10th