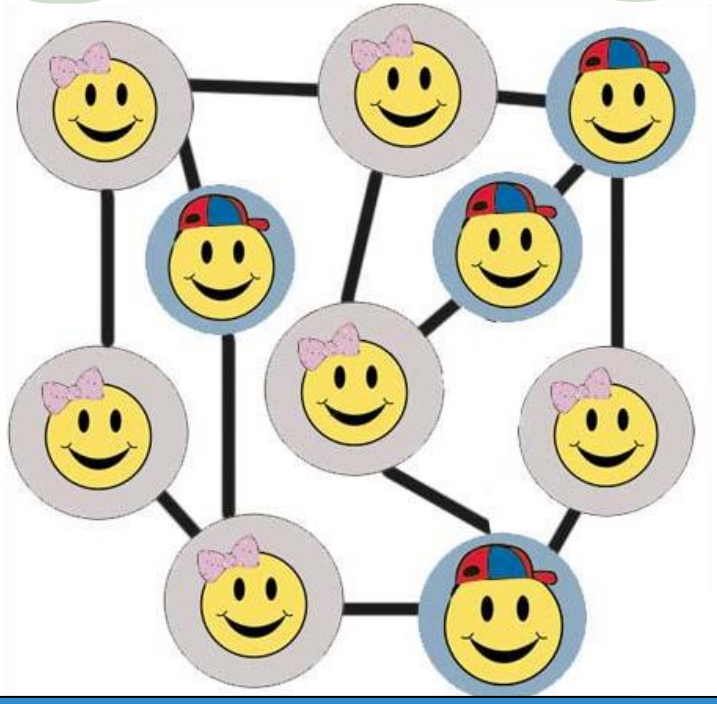
Операционные Системы и Оболочки

Одинцов Игорь Олегович
https://vk.com/os_2015

Лекция №7
13 апреля 2015 г.

Лекция 7
Средства коммуникации (продолжение)



Проблемы современных сетей

Устаревшая архитектура

- Первые сети разрабатывались в 1960-1970х
- Изменилась социальная роль и значимость компьютерных сетей в обществе

  [VoIP, потоковое видео, социальные сети]
- Меняется парадигма организации вычислений

  [на смену клиент-серверной архитектуре пришли облачные вычисления и центры обработки данных]
- Изменилась структура сети

  [число wireless пользователей превышает число wired пользователей]

Изменились требования к сети

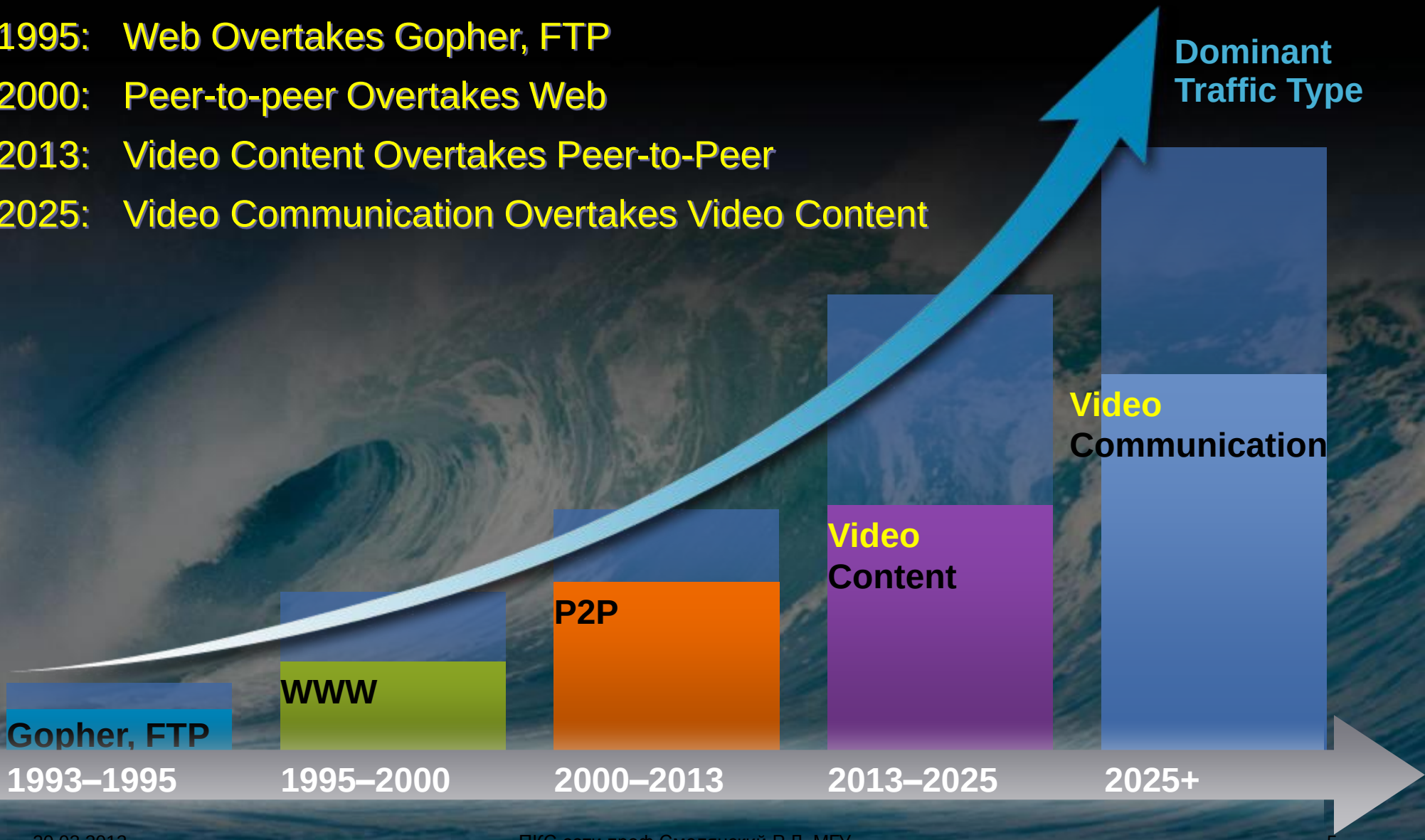
Fueled by Video

1995: Web Overtakes Gopher, FTP

2000: Peer-to-peer Overtakes Web

2013: Video Content Overtakes Peer-to-Peer

2025: Video Communication Overtakes Video Content



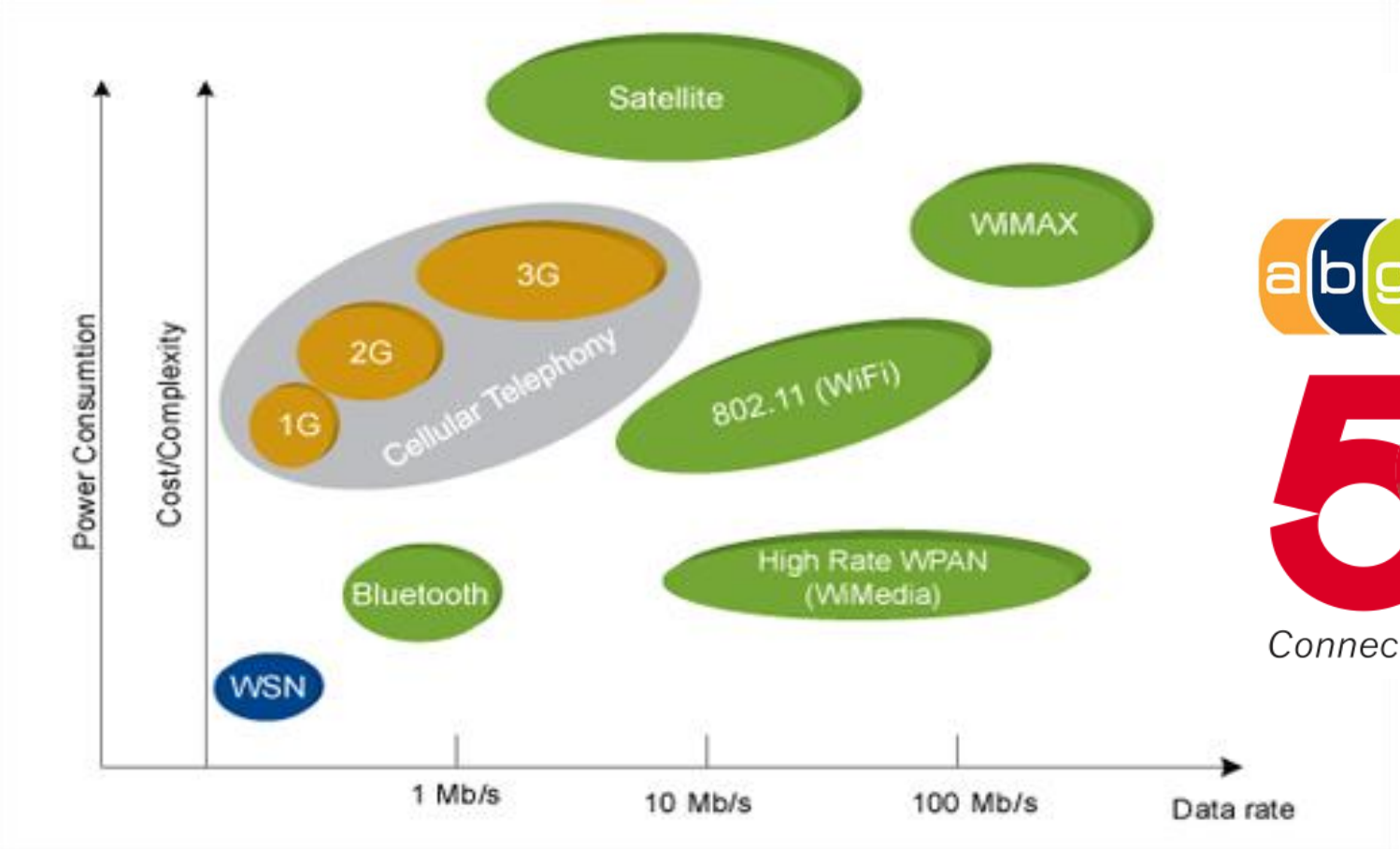
Source: Cisco Global IP Traffic Forecast

World Wide Web

Меняется парадигма организации вычислений

- На смену клиент-сервисной архитектуре пришли Cloud Computing и концепция Software as a Service
- Fog Computing

Постоянно появляются новые wireless технологии



abg WiFi n ac

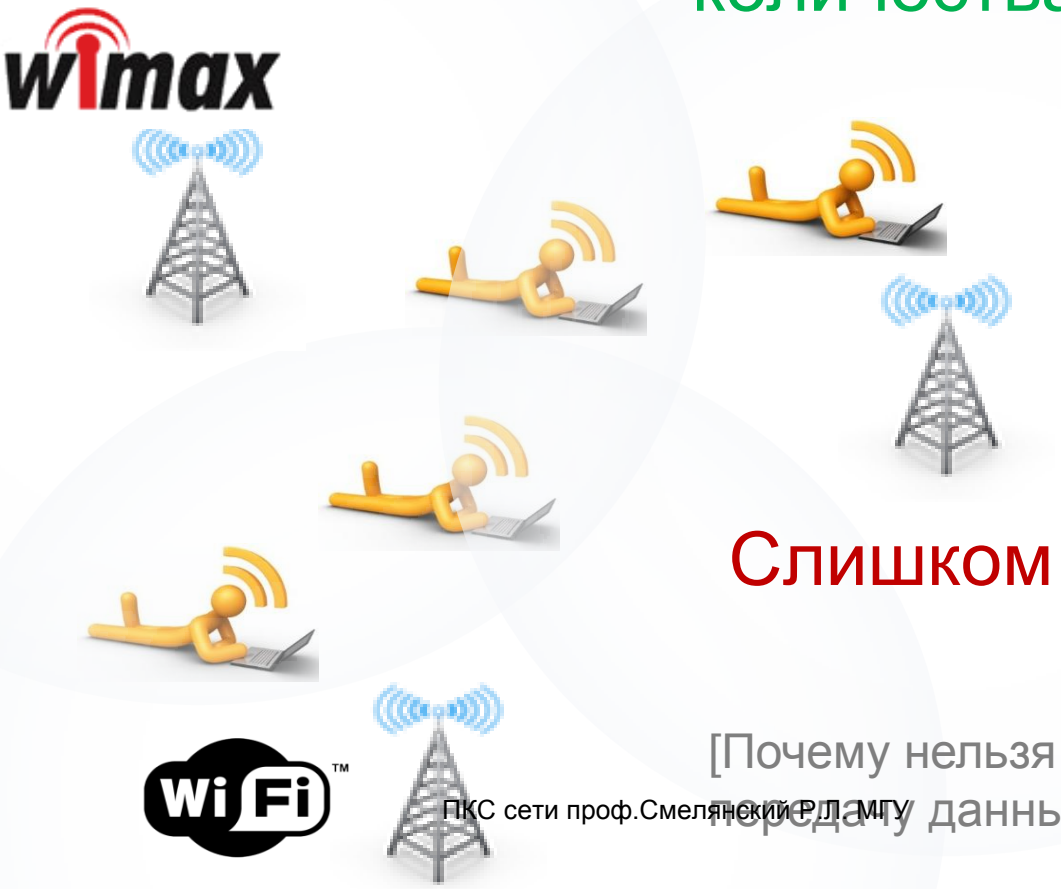
5G WiFi™

Connecting everything. Faster.™

Быстрее, дальше, дешевле!

При увеличении числа пользователей пропускная способность разделяется

Решение – увеличение количества передающих станций

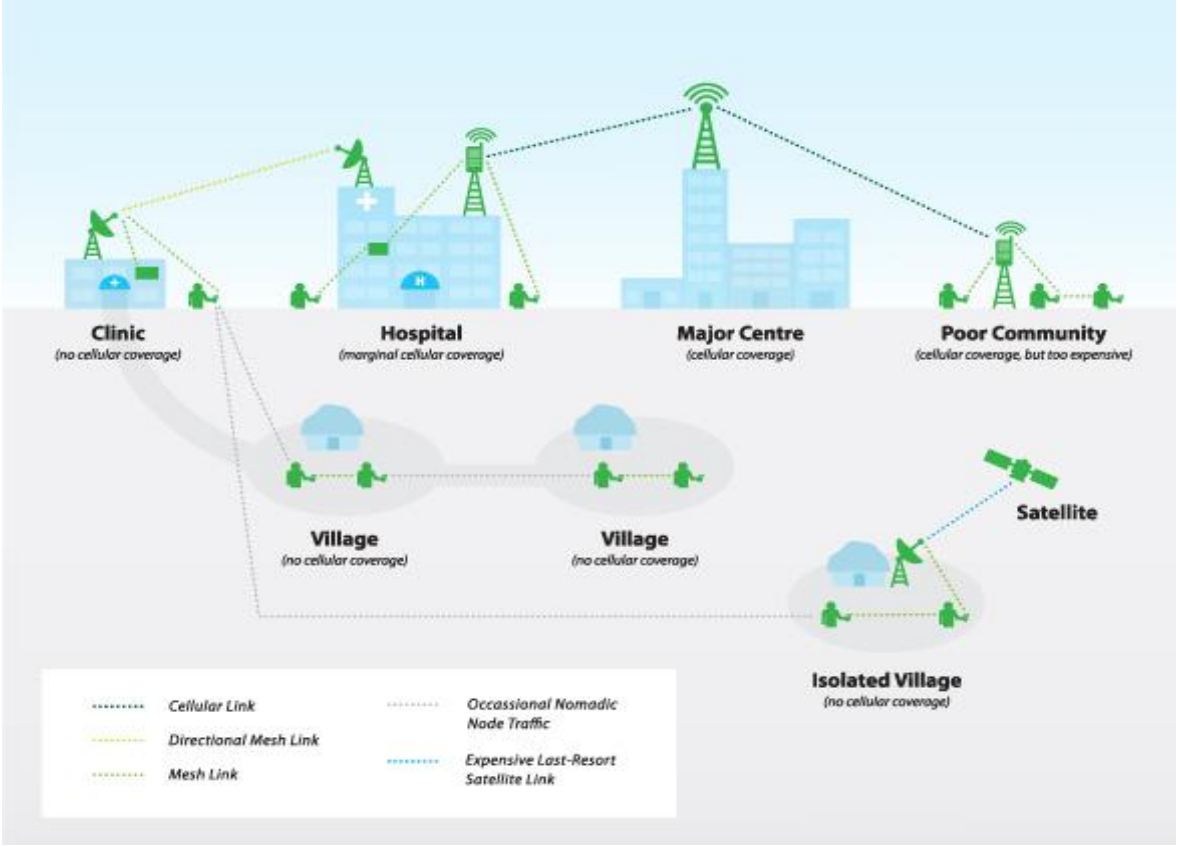


Слишком дорого!

[Почему нельзя мультиплексировать передачу данных через разные каналы?!]

Сети мобильных устройств
[Serval Project]

Идея – связать мобильные устройства с базовыми станциями транзитивно

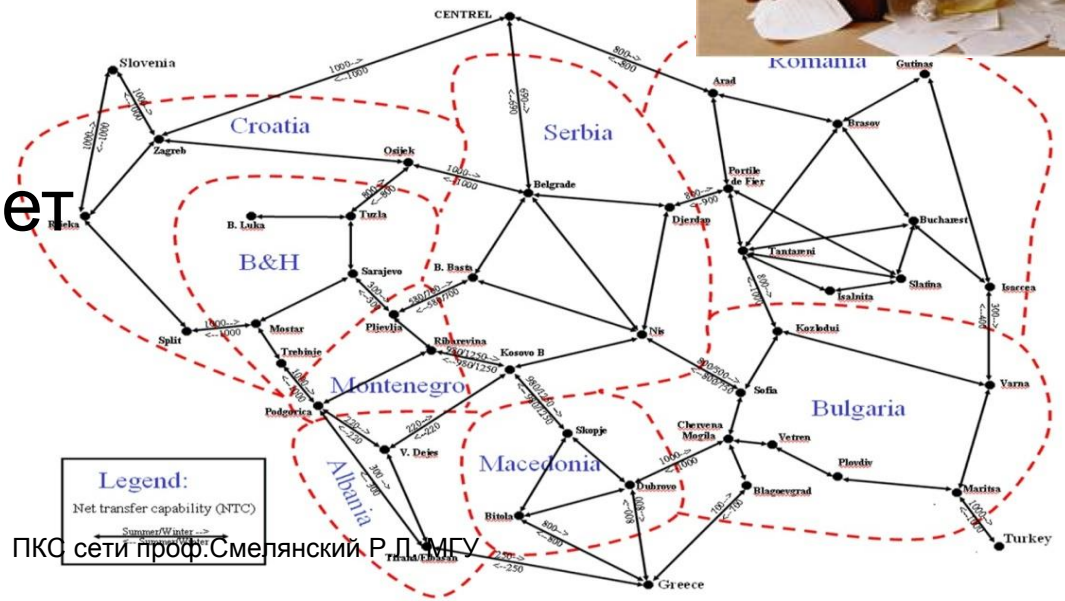


Увеличивает зону покрытия

Не может увеличить пропускную способность

Невозможность гибко управлять маршрутизацией внутри сети

- Администратор вынужден работать в терминах сетевых адресов и пакетов
- Протоколы динамической маршрутизации сложны и не всегда оптимальны
- Использование большого числа протоколов порождает множество неявных зависимостей

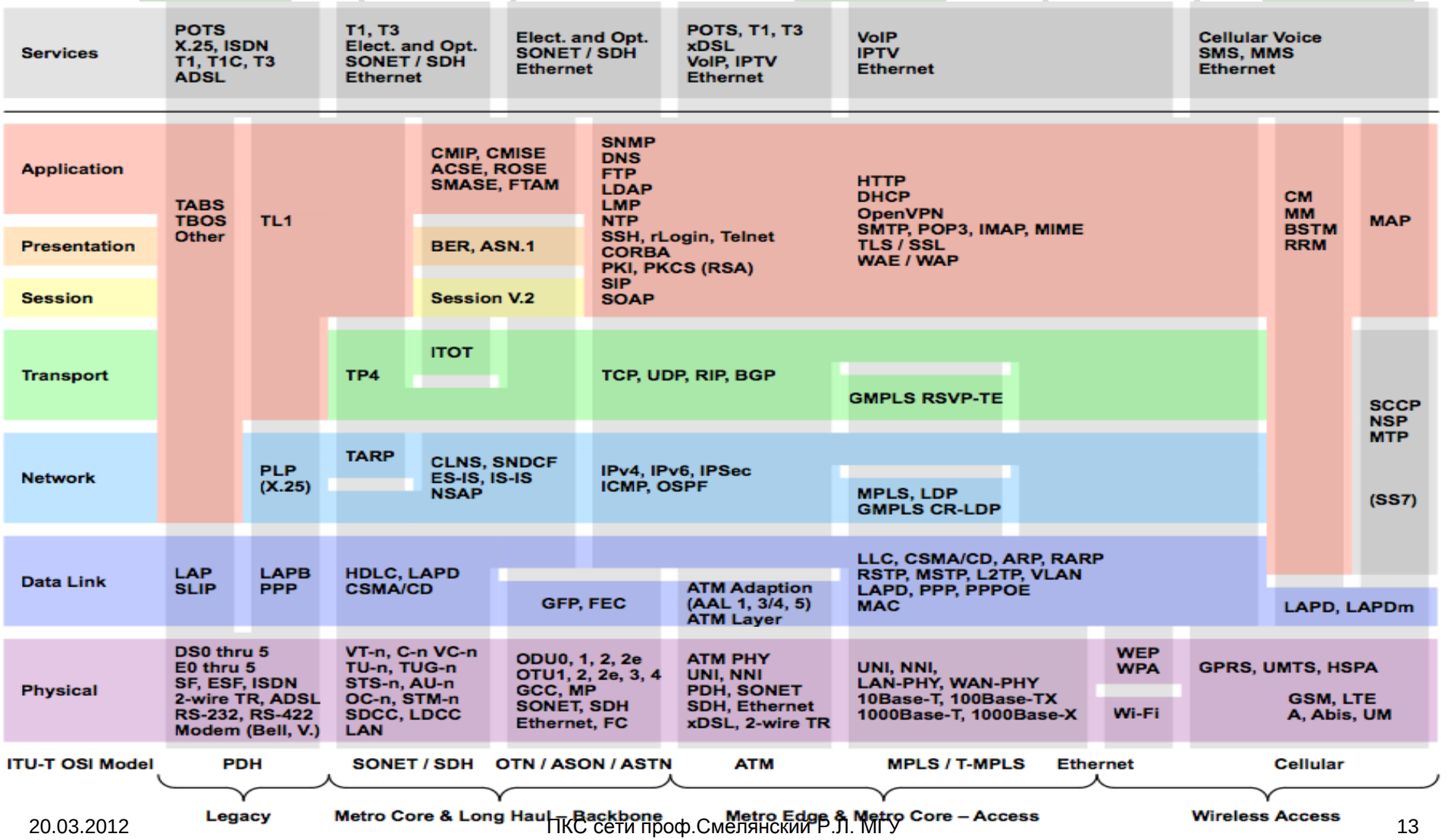


Существующая модель стека протоколов несовершенна

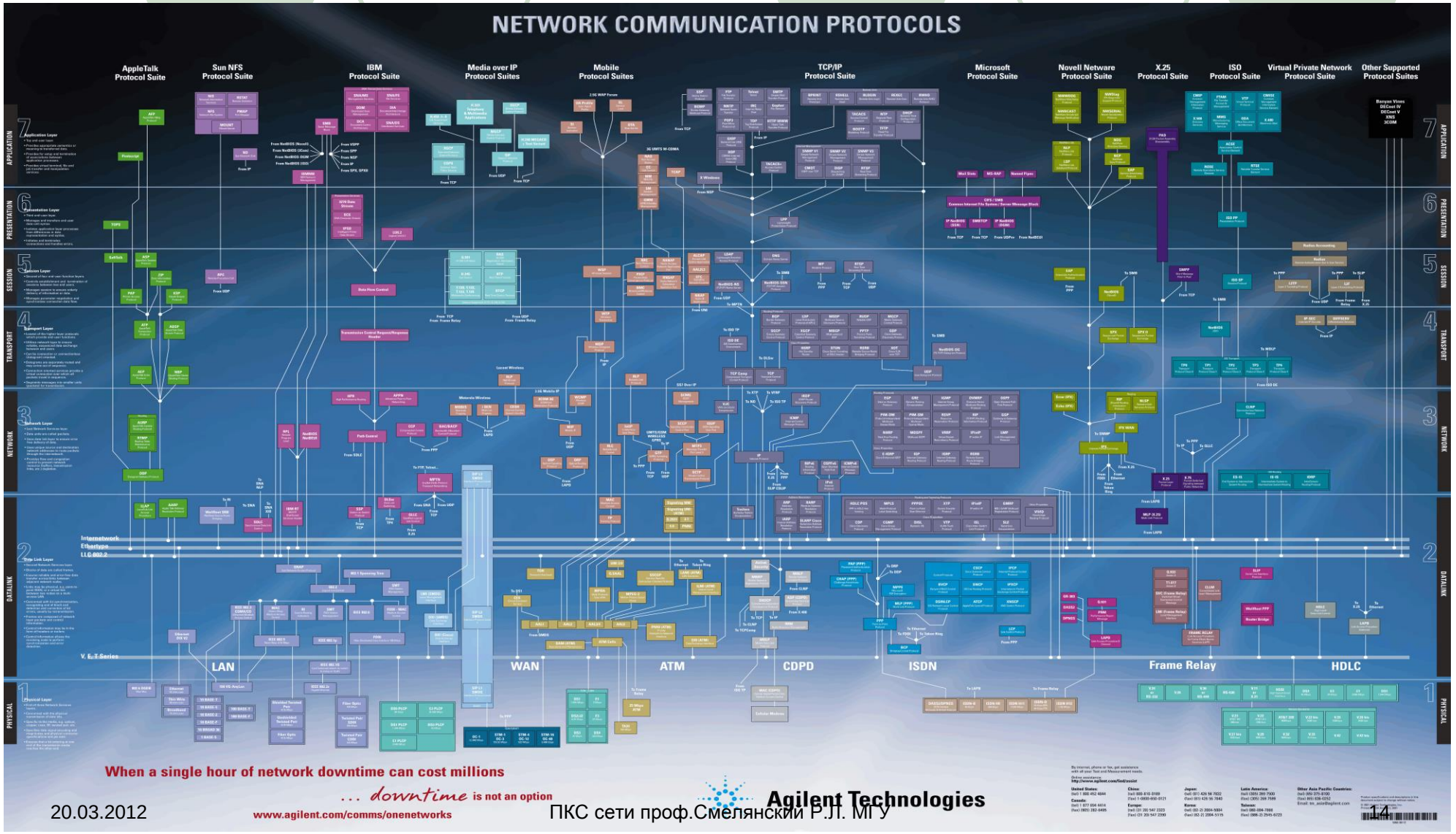
OSI Layers	OSI Layer Carries
Application	Data
Presentation	
Session	
Transport Layer	Segments
Network	Packets
Data Link	Frames
Physical	Bits

TCP/IP Layer	TCP/IP Layer Function
Application	HTTP, DNS, etc.
Transport	TCP & UDP
Internet	IP
Network Interface	Data Link
	Physical

Существующая модель стека протоколов несовершенна



Протоколы множатся



Сети закрыты для инноваций

Существует множество идей по улучшению существующих сетей

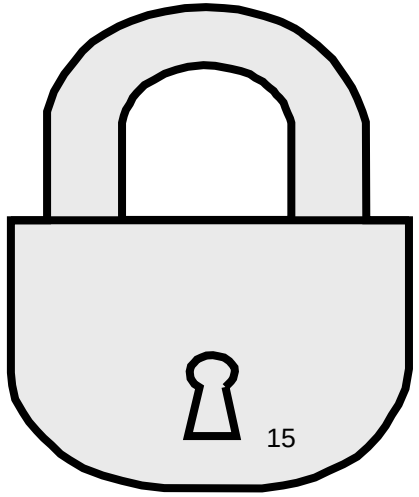
- Повышение безопасности работы в сети
- Расширение области применения сети

  [оплата проезда с мобильного телефона]

  [борьба с DDOS, поиск и устранение уязвимостей]
- Повышение эффективности работы сети

  [green switching]
- Расширение функциональности сети

  [Обеспечение необходимого качества сервиса]



Сложность проведения экспериментов

Сети закрыты для инноваций

6000+ RFC документов

Специализированное Программное Обеспечение

Специализированное устройство передачи данных

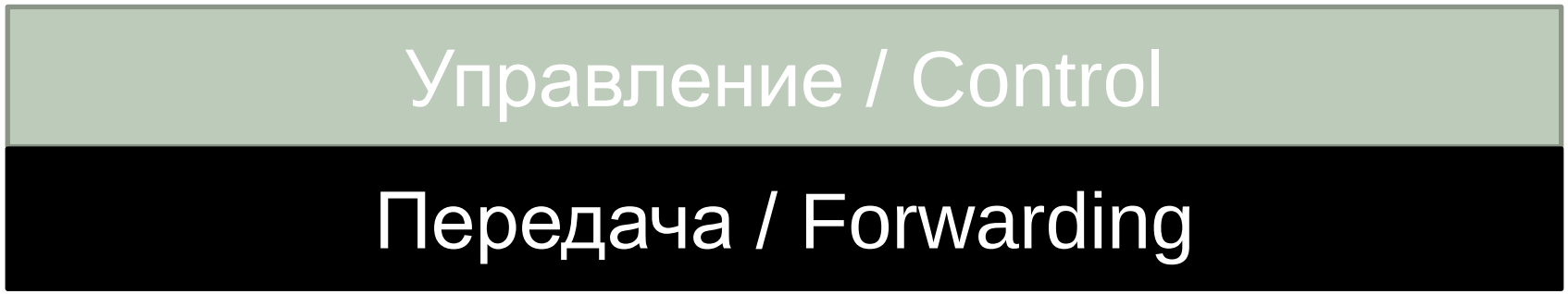
Миллионы строк закрытого проприетарного кода

Миллиарды транзисторов

Сложность внедрения новых идей

Программно-Конфигурируемые сети

[разделение передачи и управления]



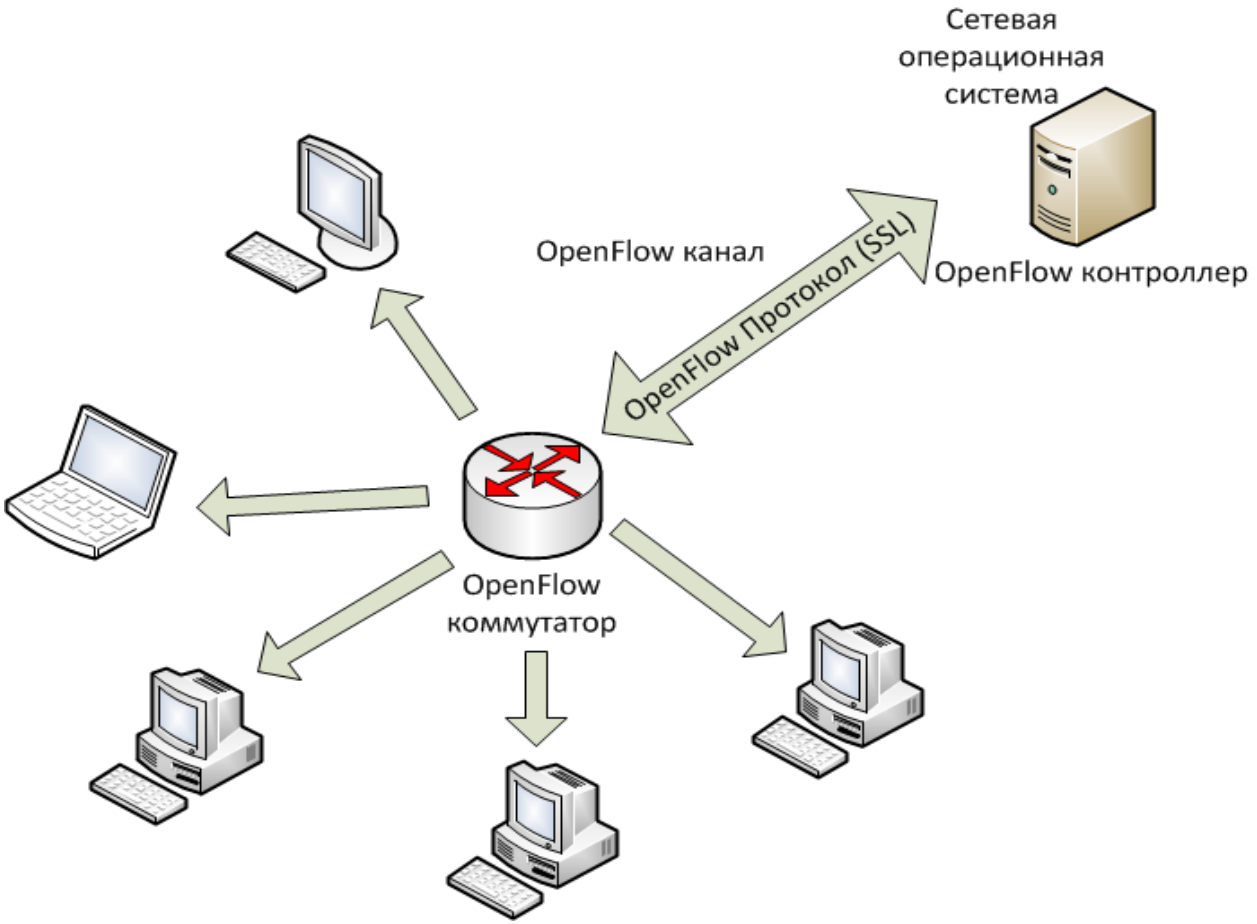
- Существует фиксированный набор простых инструкций обработки пакетов
- Концепция совместима с различными протоколами
- Работает с разноуровневым оборудованием

Программно-Конфигурируемые сети

[разделение передачи и управления]

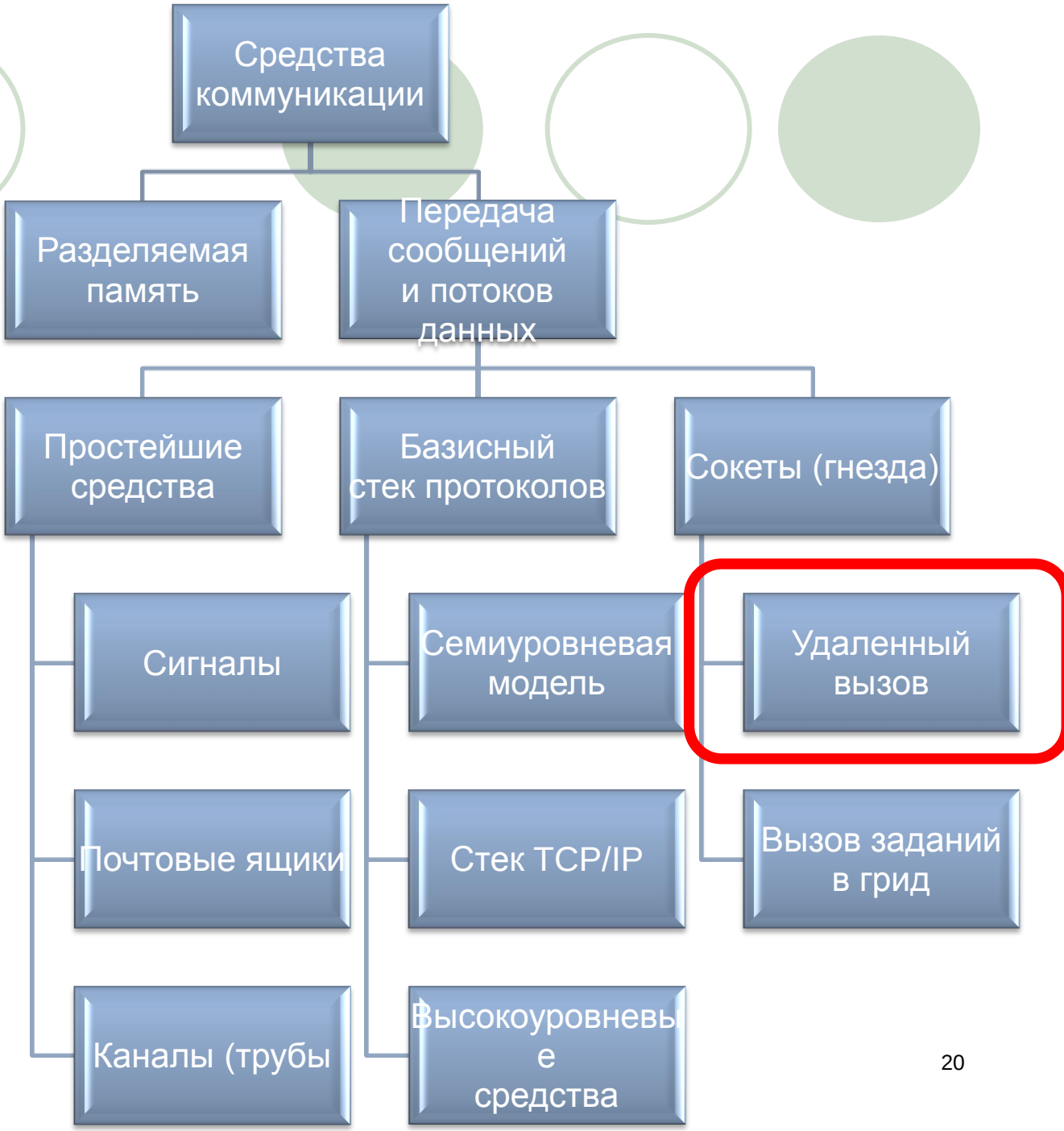
- Существует фиксированный набор простых инструкций обработки пакетов
- Концепция совместима с различными протоколами
- Работает с разноуровневым оборудованием

Технология OpenFlow



- Разделение уровней управления и передачи данных
- Управление данными с помощью контроллера

Механизмы обеспечения коммуникации



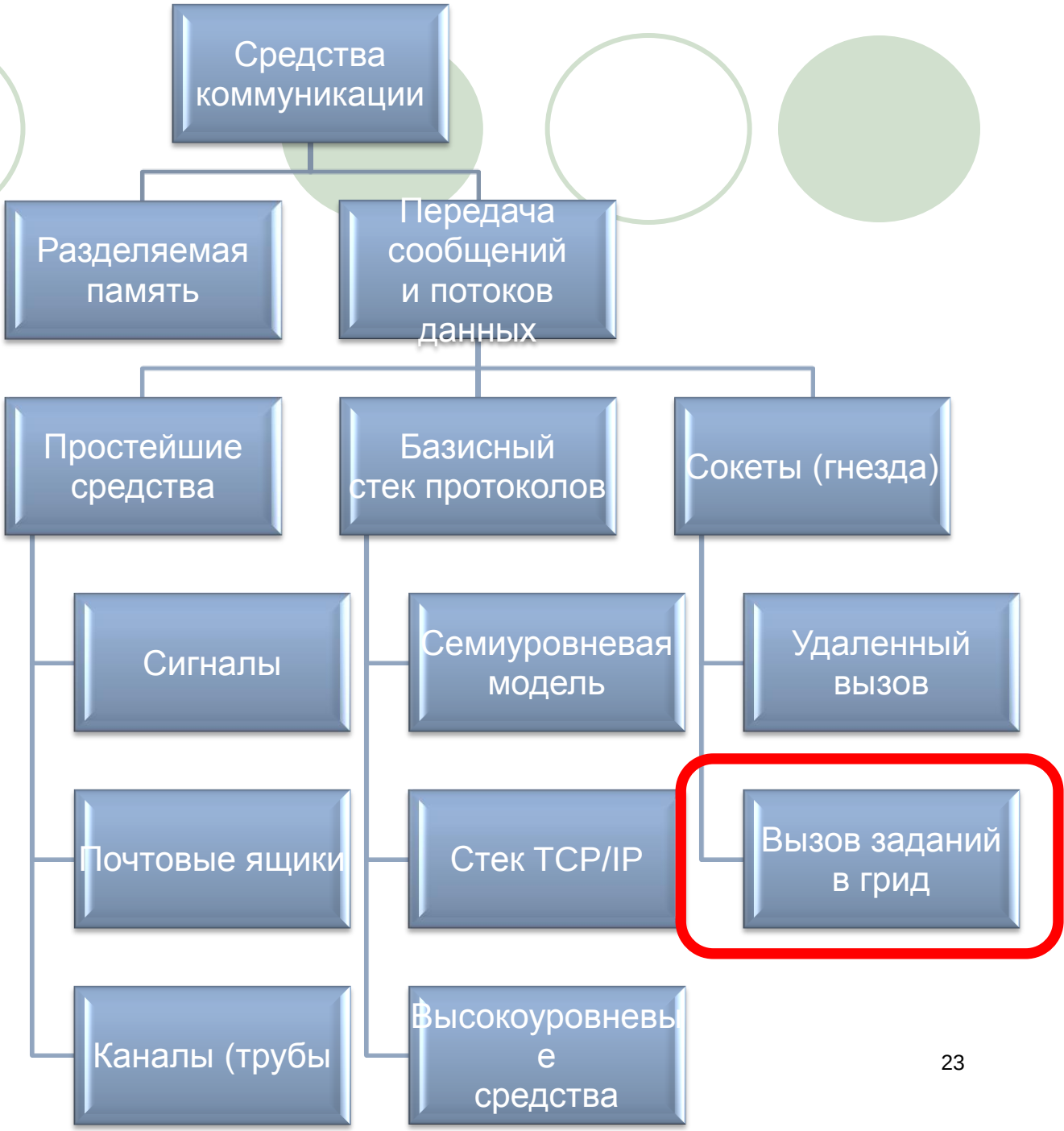
Удаленный вызов процедур

- *Удаленный вызов процедур* (Remote Procedure Call — RPC) основан на том, что процесс, исполняющийся на одном компьютере, запускает процесс на удаленном компьютере
  - Фактически осуществляется вызов процедуры, которая реально находится и поддерживается на другом компьютере
  - Удаленный вызов внешне очень похож на локальный вызов, но в нем существенно отличаются действия по передаче параметров и получению результата

Действия могут быть описаны следующим образом

- Программа-клиент производит локальный вызов процедуры, называемой "заглушкой", при этом клиенту кажется, что, вызывая "заглушку", он на самом деле вызывает процедуру сервера. В действительности, задача "заглушки" — принять аргументы, адресуемые вызываемой процедуре, преобразовать их в некоторый формат и сформировать сетевой запрос
- Сетевой запрос пересылается по сети на удаленную систему, при этом, например, используется стек протоколов TCP/IP
- На сервере все происходит в обратном порядке: "заглушка" сервера ожидает запрос и при его получении извлекает из него параметры
- "Заглушка" сервера выполняет вызов настоящей процедуры (которой адресован запрос клиента), передавая ей нужные параметры
- После выполнения процедуры при передаче результатов ее работы управление опять возвращается в "заглушку" сервера, которая формирует сообщение-отклик
- Отклик передается клиенту
- Отклик принимается "заглушкой" клиента, которая извлекает необходимые данные и передает их программе. Процесс завершен

Механизмы обеспечения коммуникации



Грид – далекий потомок RPC

- Распределенные вычисления
- Суть: объединить вычислительные ресурсы серверов, память, сети в одну большую систему с целью предоставить всю мощь этих ресурсов пользователю для решения вычислительных задач
- Цель: виртуализация ресурсов для решения проблем
- Важная характеристика: эффективность и оптимальное использование ресурсов

Что такое грид?

Грид – это среда, обеспечивающая интеграцию, виртуализацию и управление (менеджмент) сервисов и ресурсов в распределенном гетерогенном окружении, поддерживающем группы пользователей и ресурсы (виртуальные организации) в рамках традиционных административных и организационных доменов (реальные организации)
Open Grid Services Architecture Glossary of Terms
http://forge.gridforum.org/projects/ogsa-wg

- Грид-вычисления – это один из методов распределенных вычислений, базирующийся на идеях использования распределенных процессорных мощностей, приложений, данных, систем хранения и сетевых ресурсов.
  - В основу современного понимания грид-технологий ставится следующий принцип: пользователь не должен задумываться о том, на каком физическом компьютере хранятся данные и исполняются программы.
- Интерес к грид есть практически у всех лидирующих технологических компаний

Основные возможности грид

Использование распределенных ресурсов

Однократная регистрация

Безопас-ность

Грид – это среда, которая обеспечивает потребности индустрии в высокопроизводительных вычислениях и хранении данных

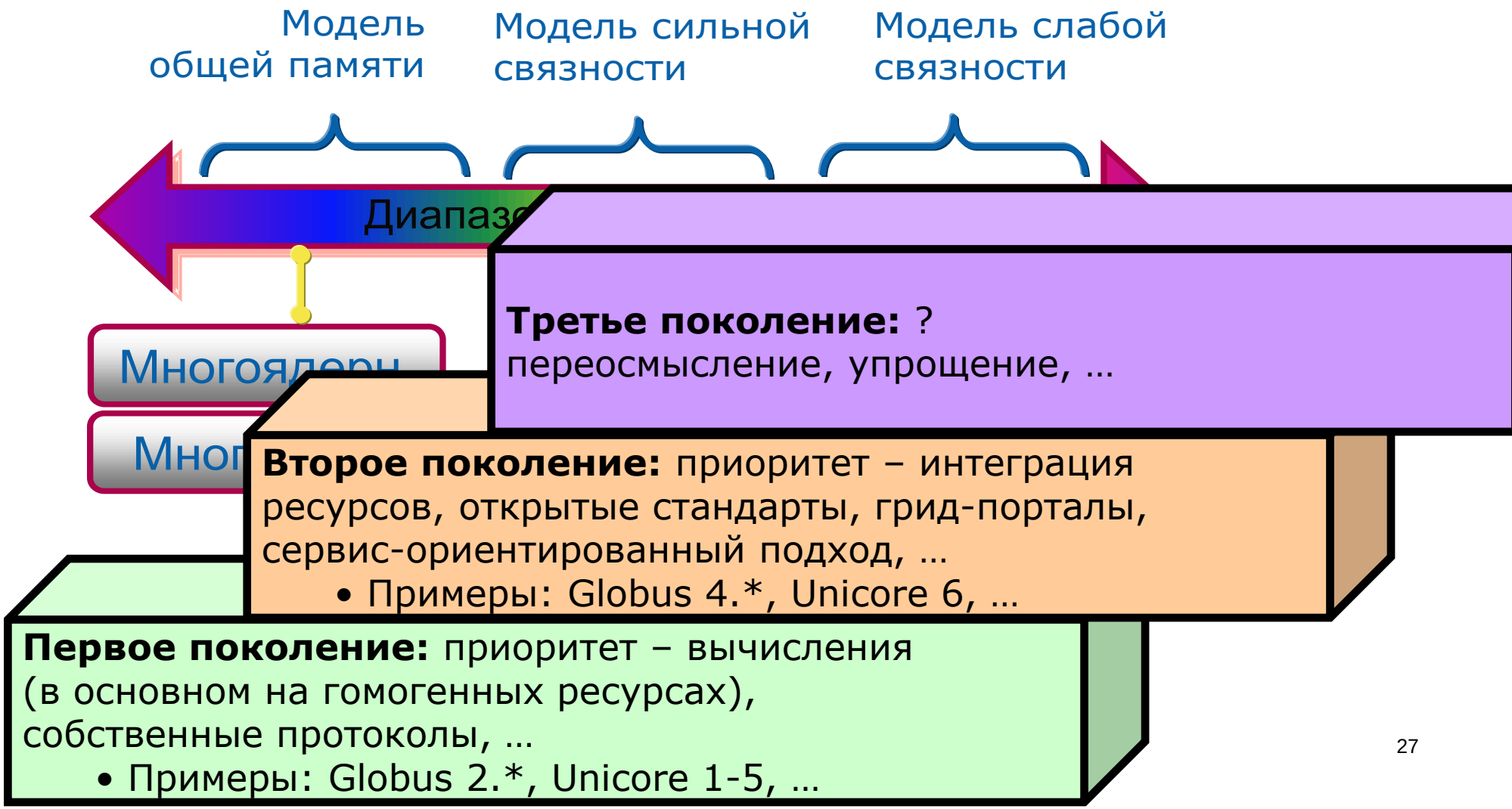
Надежность

Аппаратура и ПО

Равномерная загрузка

Открытые стандарты

Роль грид и поколения грид



Грид-среда

- Среда – инфраструктура грид в целом, состоящая из ресурсов и программных средств промежуточного уровня для обеспечения управления этими ресурсами
- Инфраструктура грид означает объединение грид-сегментов
- Программные средства промежуточного уровня – программное обеспечение, обеспечивающее согласованное управление отдельными компонентами распределенной компьютерной среды
  - ПО промежуточного уровня является прослойкой между системным и прикладным ПО

Ресурсы грид

- Ресурс грид – любая аппаратная или программная единица или объединение нескольких аппаратных или программных средств, рассматриваемых как единое целое:
  - компьютер,
  - программа,
  - хранилище данных,
  - каталог,
  - сетевой ресурс,
  - датчик,
  - распределенная файловая система,
  - компьютерный кластер,
  - и т.д.
- Заметим, что ресурс может использовать внутренние протоколы, которые не являются непосредственными компонентами грид-архитектуры
  - Например, протокол NFS или протоколы управления кластером

Электронное общество

- Виртуальной организацией (ВО) грид называется сообщество пользователей с динамически меняющимся составом, необходимое для совместного использования распределенных грид-ресурсов в рамках общих научных или технических целей и задач

Запуск типичной вычислительной задачи на грид

Workflow Engine

Задача

```
// run simulation on 100 different data sets
// retrieved from database
for(i=0; i<100; i++) {
  InputData input=getInputData(i, DatabaseXY);
  runMySimulation(input);
}
```

Запрос на ресурсы
NrOfProcessors=4
Memory=1Gb
Application=MySimulation
Policy=As_fast_as_possible

```
// create animation from output data
OutputData output = new OutputData[100];
for(i=0; i<100; i++) {
  output[i]=getOutputData(i);
}
doPostprocessing(output);
```

Запрос на ресурсы
NrOfProcessors=16
Memory=4Gb
Application=SimViz 4.0
Policy=As_fast_as_possible

Запуск типичной вычислительной задачи на грид

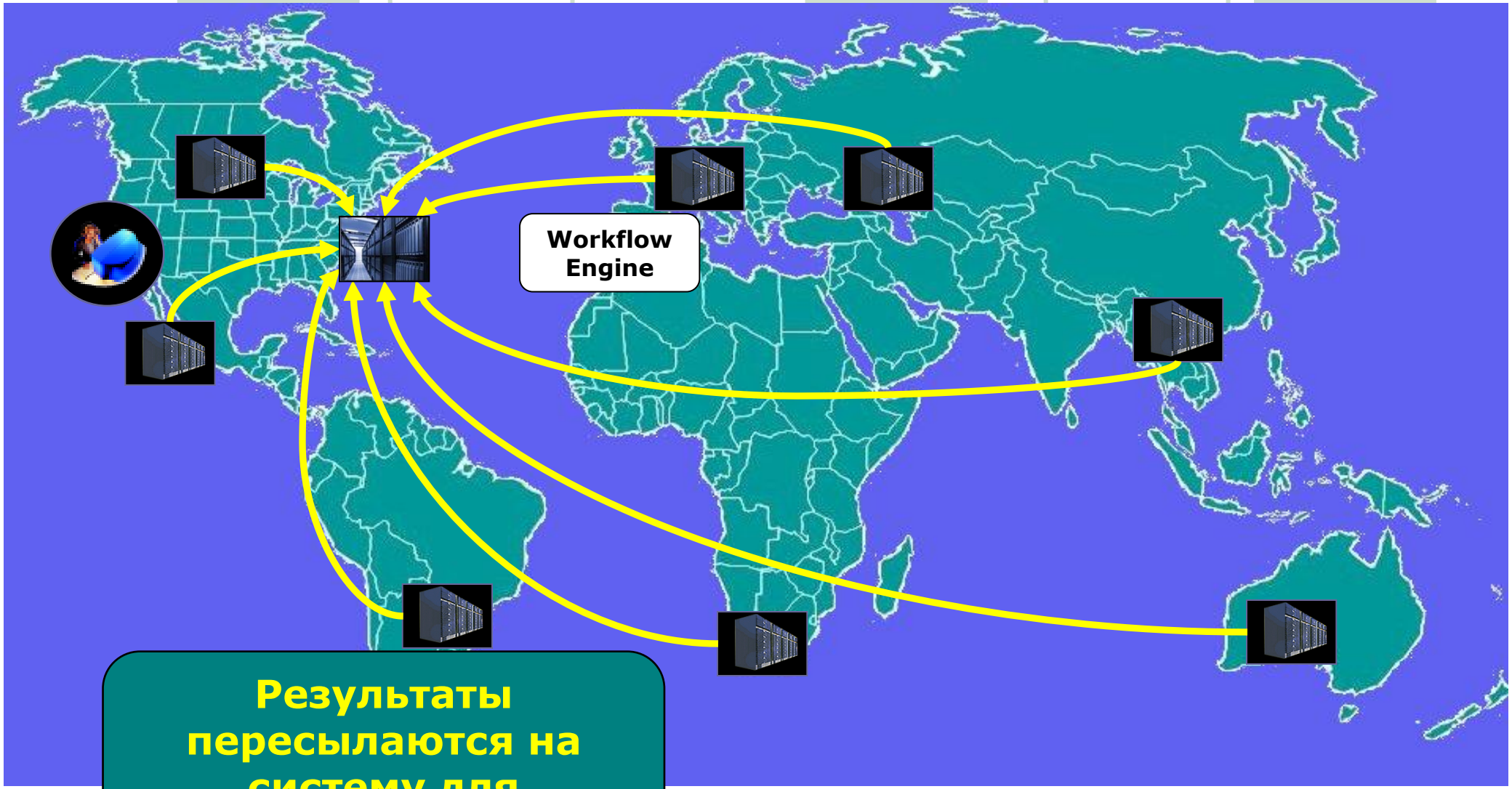
Запуск типичной вычислительной задачи на грид

Запуск подзадач на доступных целевых системах

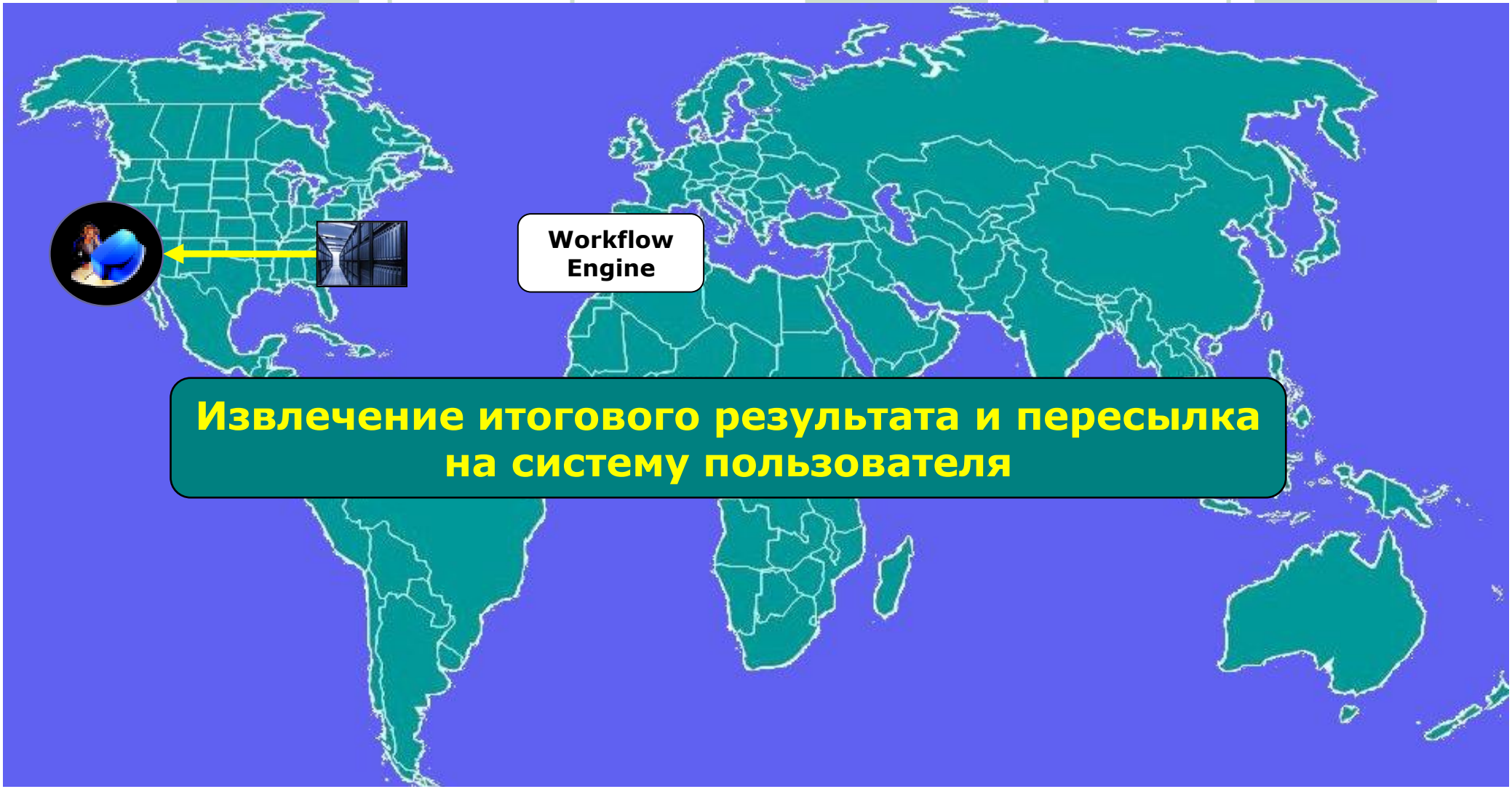
Запуск типичной вычислительной задачи на грид

Подзадачи запрашивают исходные данные из базы данных

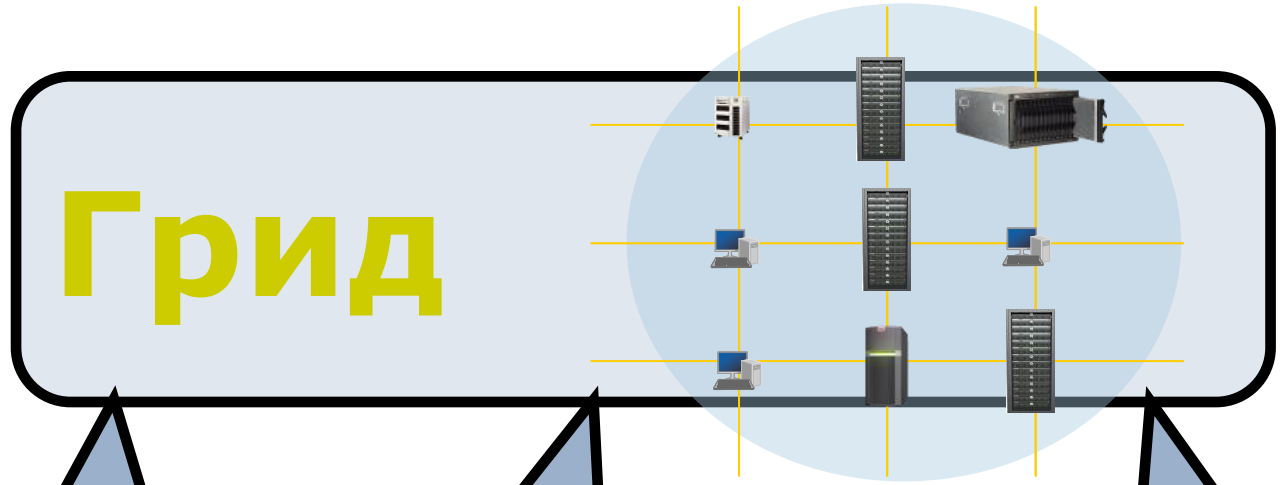
Запуск типичной вычислительной задачи на грид



Запуск типичной вычислительной задачи на грид



Три технологических «источника» грид



Высоко-производительные вычисления

Сервис-ориентированная архитектура

Виртуализация

Место грид в высокопроизводительных вычислениях (HPC)

Кластеры	Гриды
Сильно-связная архитектура: локальные узлы, скоростная связь между узлами	Слабо-связная архитектура: распределенные и удаленные узлы, LAN/WAN/Internet
Гомогенная архитектура	Гетерогенная архитектура
Явное предоставление архитектуры и ресурсов	Детали архитектуры скрыты, виртуализация ресурсов
Управляется как единое целое, одна организация	Распределенное управление, несколько организаций
Статическое распределение ресурсов	Динамический поиск ресурсов
Не требует повышенных мер обеспечения безопасности, поскольку находится в защищенной сети	Обеспечение целостности и конфиденциальности информации, устойчивость к сетевым атакам

Развитие грид

Кластеры

- Параллельность вычислений
- Управляются как единая система

- Управление
- Параллельные приложения
- Диспетчеризация работ

- Управление ресурсами
- Приложения, основ. на сервисах
- Инструментовка

Интра-грид

- Распределенные системы
- Ограничены рамками организации (единый домен безопасности)

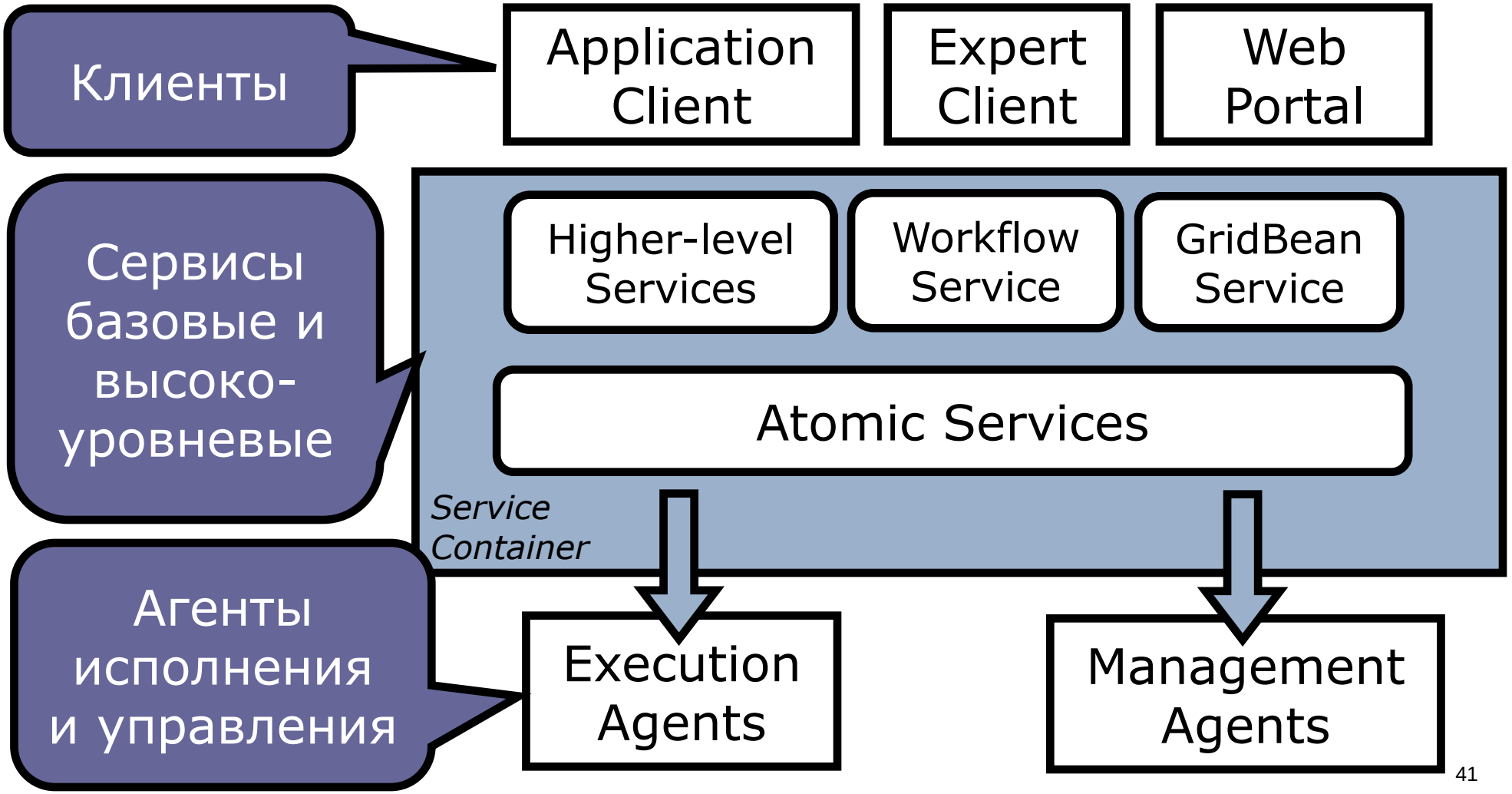
Интер-грид

- Включает различные организации

- Безопасность
- Бизнес-модели
- Архитектуры, основанные на сервисах

Инструментарий грид

Типичная архитектура грид



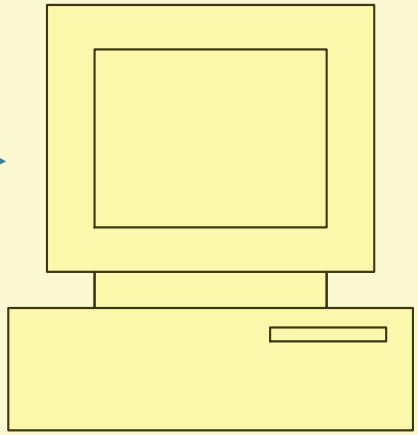
Архитектура грид с точки зрения пользователей

Клиентская сторона

GPE Clients

SOAP

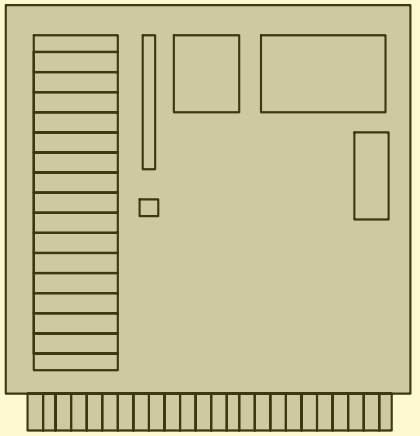
Компьютер, представляющий грид-узел в Интернет



Контейнер с веб-сервисами

RMI

Вычислительный узел (Целевая система)



Execution Agent (TSI)

Основные задачи Грид

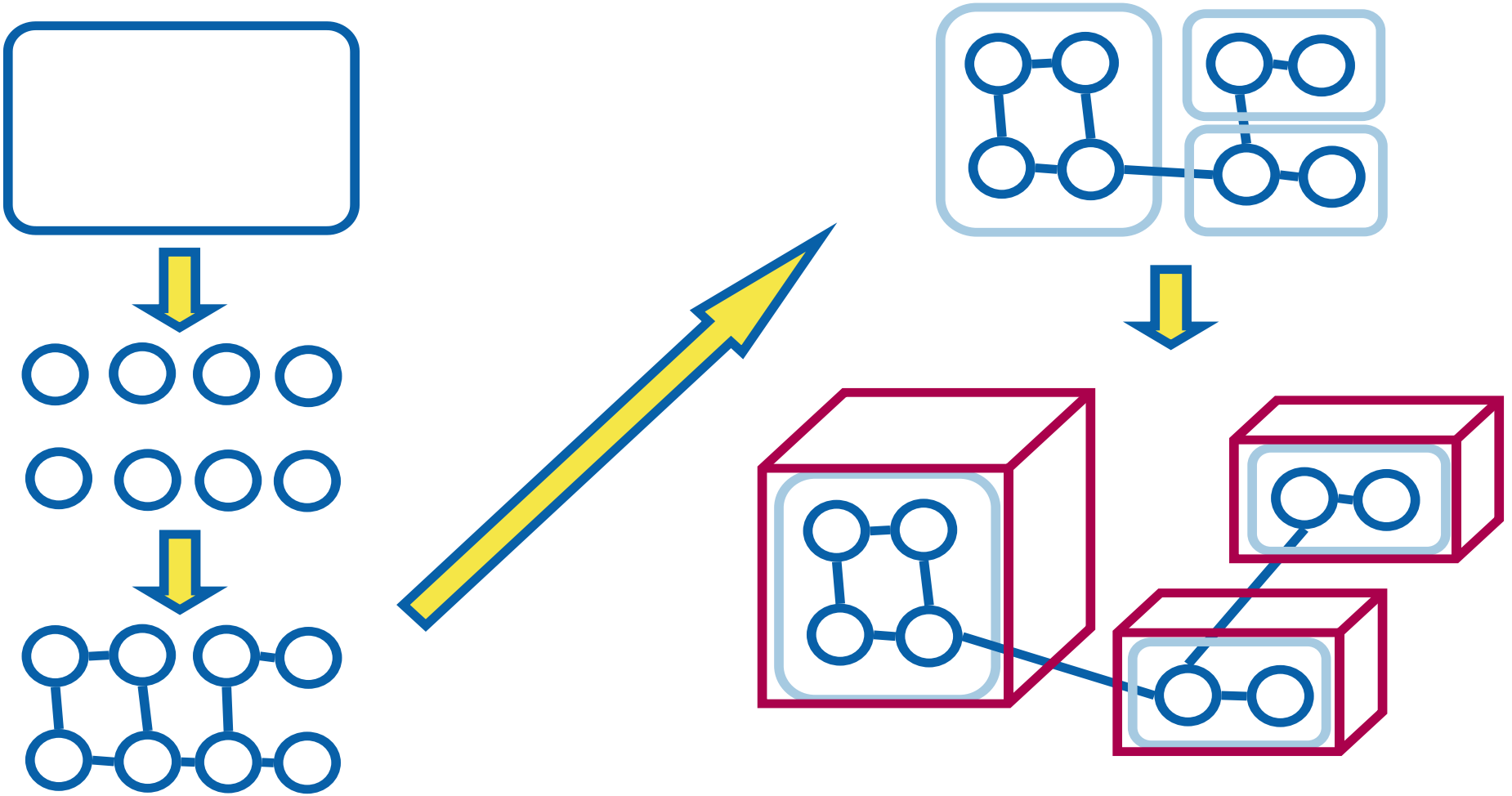
- Объединить гетерогенные ресурсы
- Предоставить доступ пользователям из любой точки мира
- Обеспечить качественное, надежное обслуживание с гарантированной производительностью
- Поддерживать универсальность среды (по приложениям, сервисам, ресурсам)

Необходимость ГРИД для БАК

- Объём получаемых данных LHC соответствует 20 миллионам записанных CD дисков в год. Где их хранить?
- Анализ данных LHC потребует вычислительных мощностей, эквивалентных мощности 100000 самых современных процессоров. Где их взять?
- Ресурсы ЦЕРН уже сейчас составляют более 1000 2-х процессорных ПК и 1 Пб памяти на дисках и на лентах. Но этого мало!!!

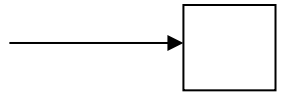
Выход – объединение вычислительных ресурсов физиков всего мира

Правильно приготовленная задача

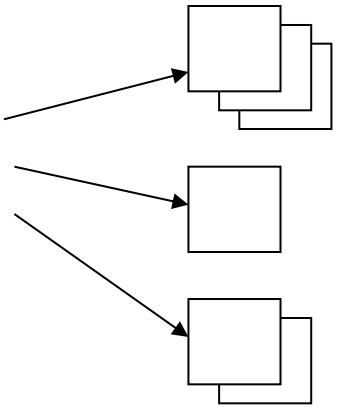


Типичные грид-приложения

- 1. Приложения, использующие ресурсы, не доступные на локальной системе

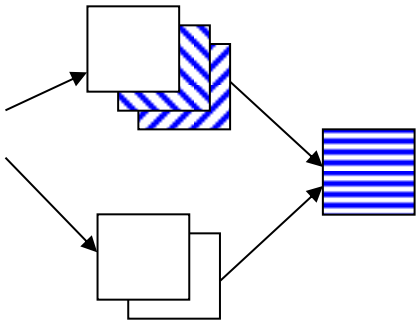


- 2. Параметрические приложения

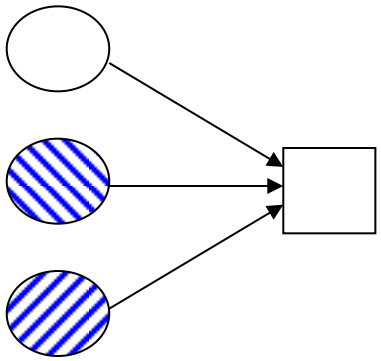


Типичные грид-приложения

- 3. Приложения, основанные на потоках

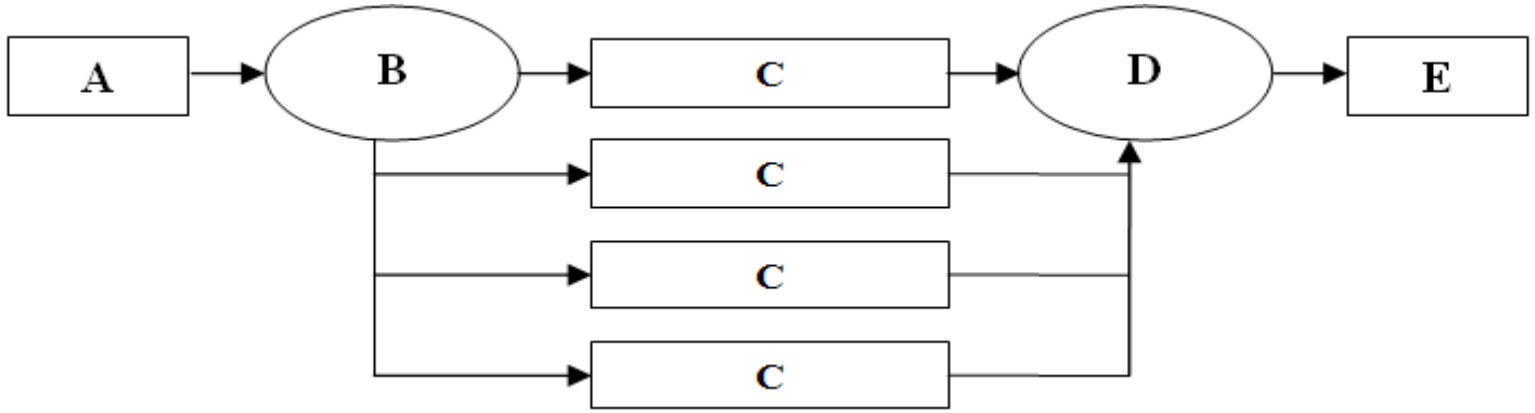


- 4. Приложения с большим объемом входной информации (Data Mining, Modeling and large Data Warehousing)



Типичные грид-приложения (крупная гранулируемость)

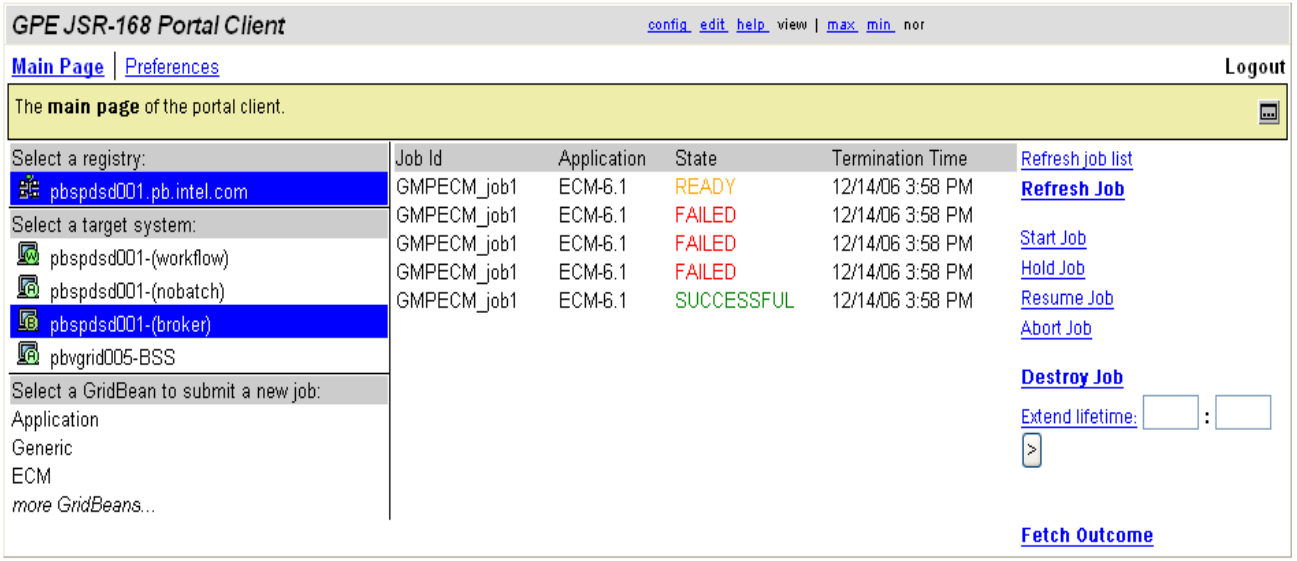
Пример: Статистическое обобщение данных по волновому климату океанов и морей – классическое приложение для распределенных вычислений, ориентированное на обработку однородных выборок сверхбольших объемов данных.



Типичные грид-приложения (независимые вычисления)

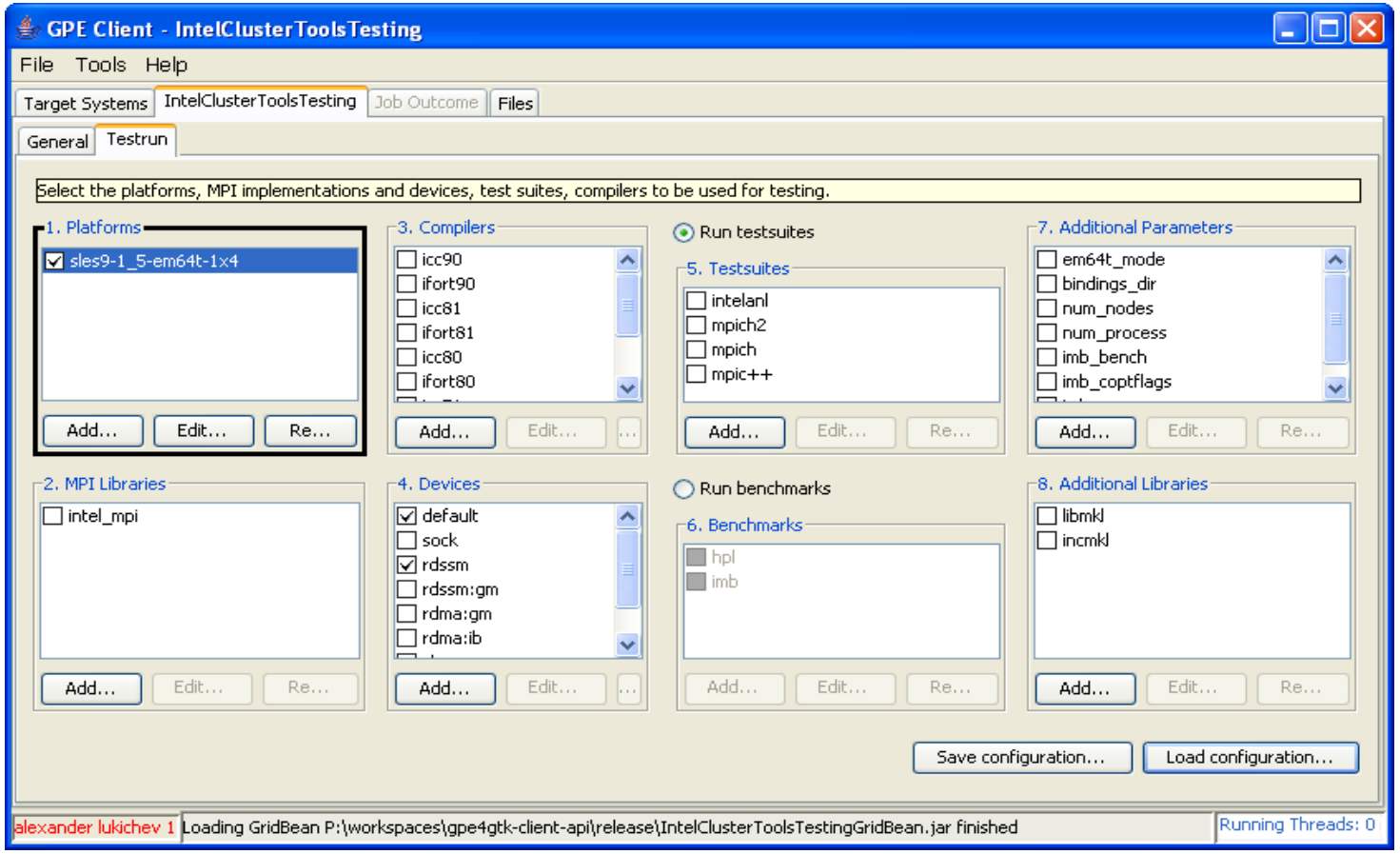
Пример: Решение задачи факторизации целых чисел

- Для любого целого $n \geq 0$: $n = p_0^{k_1} \ldots p_m^{k_m}$, где p_i – простые
  - Необходимо найти разбиение n.



RSA – один из наиболее популярных алгоритмов ассиметричного шифрования.
Основная часть ключа – modulus:
n=pq, где p,q – различные простые числа
n входит в открытый ключ, используется для шифрования
Для расшифровки необходимо знание p и q
Криптографическая стойкость обеспечивается трудоемкостью *факторизации* n.

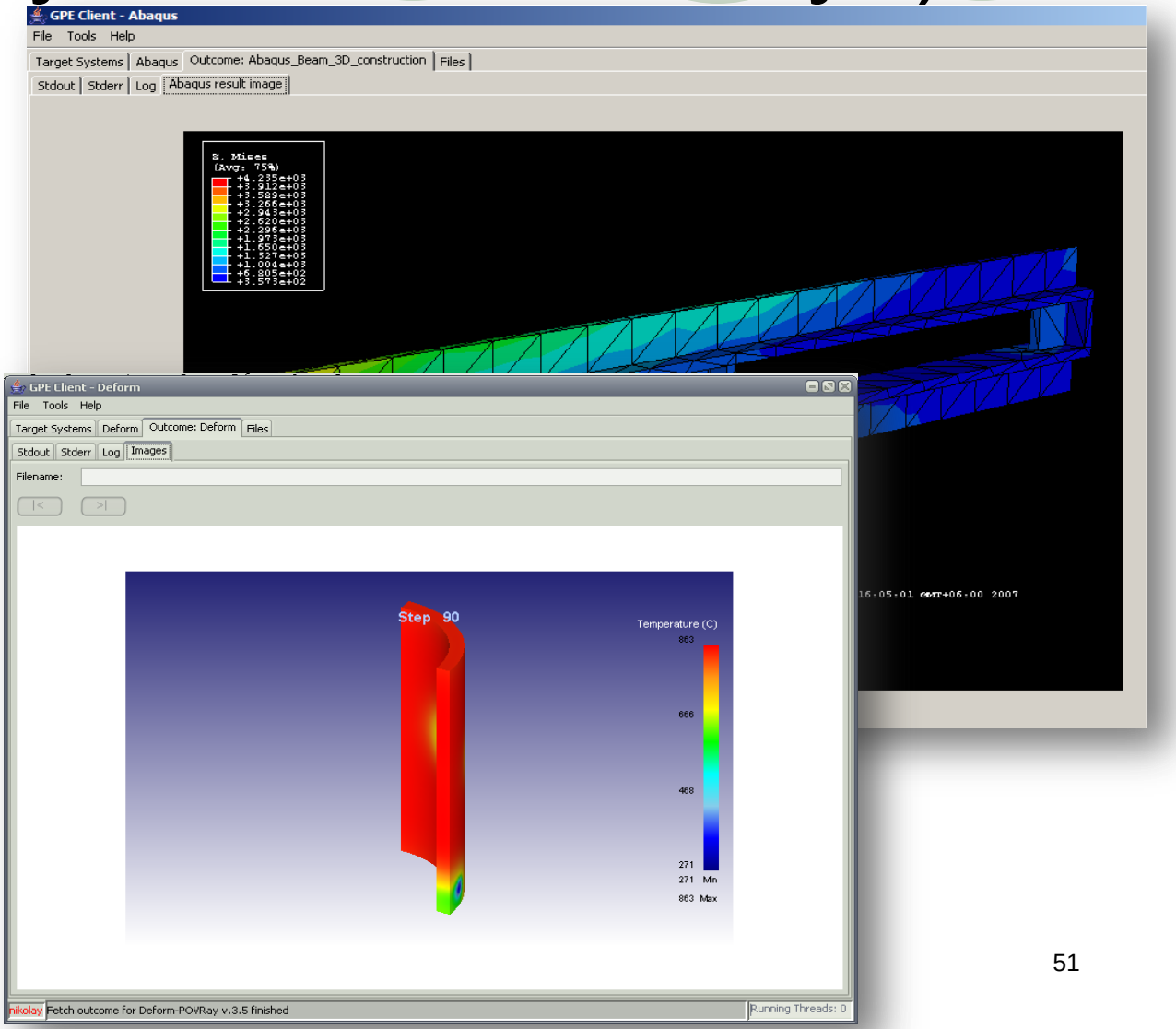
Типичные грид-приложения (интеграция, многоплатформенность)



- Пример: Тестирование библиотеки MPI

Типичные грид-приложения (интеграция, удаленный доступ)

- Пример: CAEBeans - иерархические системы структурированных проблемно-ориентированных оболочек над инженерными пакетами



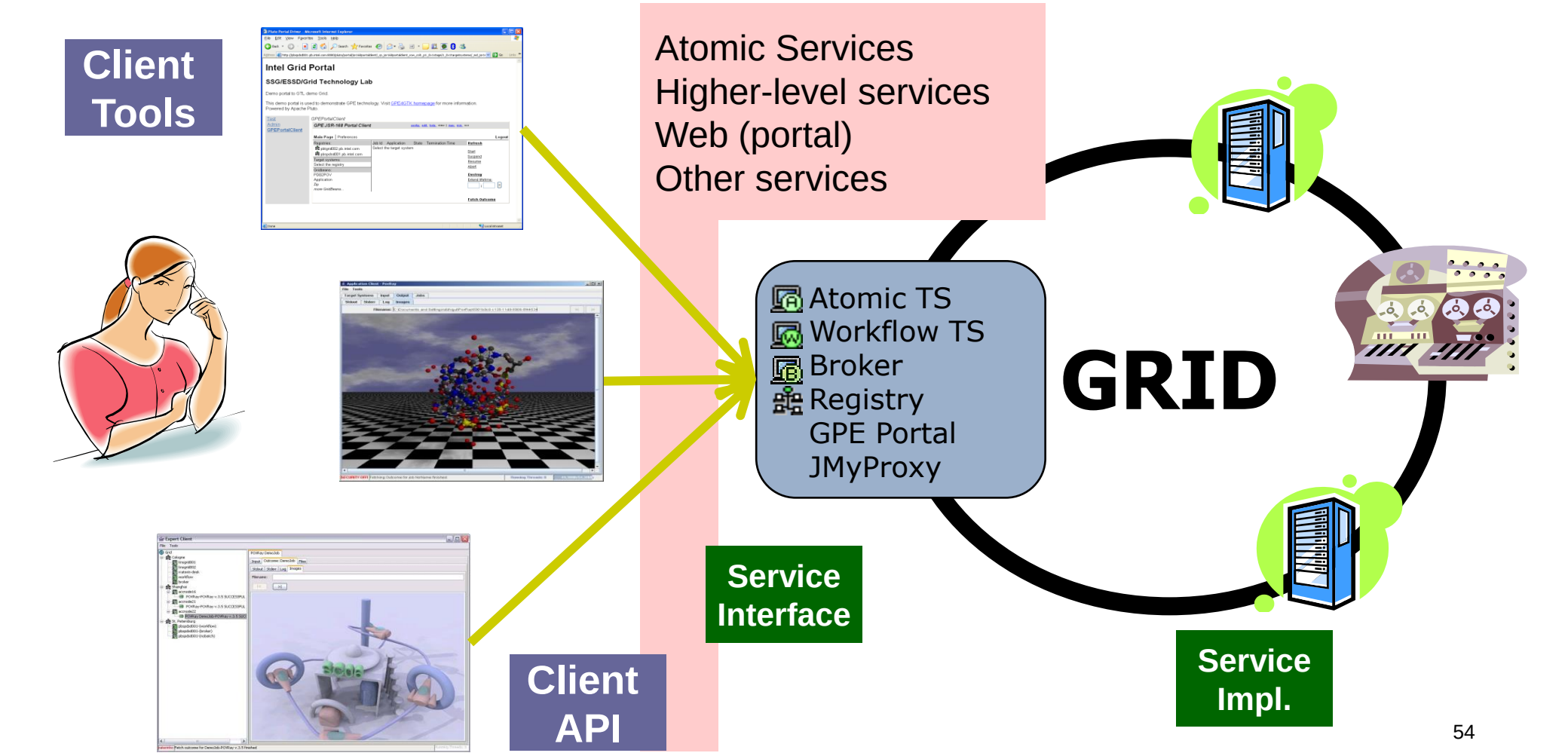
Области применения грид

- Изначально грид-технологии предназначались для решения сложных научных, производственных и инженерных задач, которые невозможно решить в разумные сроки на отдельных вычислительных установках
- По мере развития грид проникает в промышленность и бизнес, крупные предприятия создают грид для решения собственных производственных задач
- В результате грид претендует на роль универсальной инфраструктуры для обработки данных, в которой функционирует множество служб (Grid Services).
  - Службы позволяют решать не только конкретные прикладные задачи, но и предлагают сервисные услуги: поиск необходимых ресурсов, сбор информации о состоянии ресурсов, хранение и доставка данных
- Грид успешно применяется для решения следующих классов задач:
  - **Массовая обработка потоков данных большого объема**
  - **Многопараметрический анализ данных**
  - **Моделирование на удаленных суперкомпьютерах**
  - **Реалистичная визуализация больших наборов данных**
  - **Сложные бизнес-приложения с большими объемами вычислений**

Характерные черты современных грид-систем

- Основаны на веб-сервисах (и WSRF)
- Большое внимание уделяется вопросам безопасности и разграничения доступа
  - поддержка виртуальных организаций
- Применяются не только для решения сложных вычислительных задач, но и для бизнес-приложений
  - распределенный доступ к базам данных

Компоненты грид-среды с точки зрения пользователя

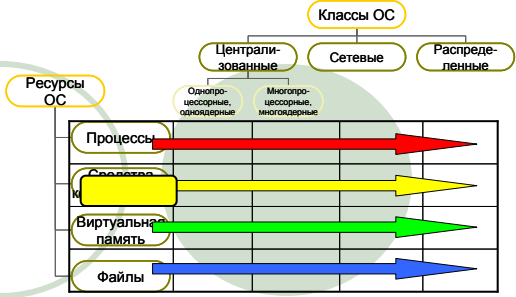


Лекция 7
Алгоритмы синхронизации

Будем строить курс вокруг **идей!**

Синхронизация

План лекции



- Средства коммуникации
- Алгоритмы синхронизации
  - Синхронизация в централизованных ОС
  - Синхронизация в сетевых и распределенных ОС
- Проблемы при работе с ресурсами

Синхронизация процессов в централизованных архитектурах

- Процессы называются *параллельными*, если они существуют одновременно
- Могут быть:
  - Независимые
  - Взаимодействующие и нуждающимися в синхронизации
- *Синхронизация процессов* — использование специальных атомарных операций для осуществления взаимодействия между процессами

Критические ресурсы и участки

- *Критический ресурс* — это ресурс, допускающий обслуживание только одного процесса за один раз. Если несколько процессов хотят использовать критический ресурс в режиме разделения, то им следует синхронизировать свои действия, чтобы ресурс всегда находился в распоряжении не более чем одного из них
  - Типичным примером критического ресурса является разделяемая переменная, суммирующая некоторую величину (назовем ее счетчик)

- *Критические участки* — это участки процесса, где происходит обращение к критическому ресурсу. Критические участки должны быть *взаимоисключаемыми*, т. е. в каждый момент времени не более чем один процесс может быть занят выполнением своего критического относительно некоторого ресурса участка. Обеспечение (поддержка механизма) взаимоисключения — ключевая задача параллельного программирования
  - Критические участки процессов для работы с ресурсом счетчик могут содержать следующий код:

```
счетчик := счетчик + 1
```

Блокировка

- *Блокировка* — предотвращение выполнения кем-либо чего-либо. Процесс должен устанавливать блокировку перед входом в критический участок и снимать ее после выхода. Естественно, если участок заблокирован, то другой процесс должен ждать снятия блокировки
- *Вход взаимоисключения* и *выход взаимоисключения* — участки процесса, обрамляющие критический участок и служащие для обеспечения взаимоисключения. Примером входа взаимоисключения может служить блокирование, а выхода — разблокирование

Задача взаимного исключения

- Необходимо согласовать работу n>1 параллельных потоков при использовании некоторого критического ресурса таким образом, чтобы удовлетворить следующим требованиям:
  - одновременно внутри критической секции должно находиться не более одного потока
  - критические секции не должны иметь приоритеты в отношении друг друга
  - остановка какого-либо процесса вне его критической секции не должна влиять на дальнейшую работу процессов по использованию критического ресурса
  - решение о вхождении потоков в их критические секции при одинаковом времени поступления запросов на такое вхождение и равной приоритетности потоков не откладывается на неопределенный срок, а является конечным во времени
  - относительные скорости развития потоков неизвестны и произвольны
  - любой поток может переходить в любое состояние, отличное от активного, вне пределов своей критической секции
  - освобождение критического ресурса и выход из критической секции должны быть произведены потоком, использующим критический ресурс, за конечное время

Пример: слишком много молока

- Как решать проблему?
  - Оставлять записку (== блокировка)
  - Убирать записку
  - Не покупать, если есть записка

```
if ( no_milk ) {
    if ( no_note ) {
        leave note;
        buy milk;
        remove note;
    }
}
```

Пример: слишком много молока

- Использование блокировок - в принципе правильный подход, но их надо правильно реализовать
- В конечном итоге получим что-то типа:

```
prog_lock();
if ( no_milk )
    buy milk;
prog_unlock();
```

- Проанализируем программные способы решения проблемы…

Алгоритм Деккера

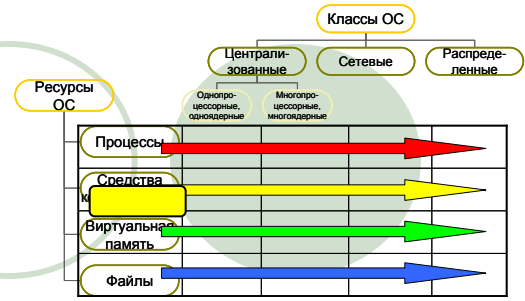
```
enum state { UNLOCKED, LOCKED };
typedef struct {
    char status[2];             /* байт статуса для каждого из двух процессов */
    char turn;                  /* какой из процессов будет следующим */
} lock_t;
void init_lock(lock_t* lock) {
    lock->status[0] = UNLOCKED;
    lock->status[1] = UNLOCKED;
    lock->turn = 0;
}
void d_lock(volatile lock_t* lock) {
    /* устанавливаем блокировку для текущего процесса */
    lock->status[cur_proc_id()] = LOCKED;
    /* проверяем, не установлена ли блокировка другим процессом */
    while ( lock->status[other_proc_id()] == LOCKED) {
        /* если другой процесс уже установил блокировку,
           проверяем — чья очередь войти в критический участок */
        if ( lock->turn != cur_proc_id()) {
            lock->status[cur_proc_id()] = UNLOCKED;
            while ( lock->turn == other_proc_id())
                    ;
            lock->status[cur_proc_id()] = LOCKED;
        }
    }
}
void d_unlock(lock_t* lock) {
    lock->status[cur_proc_id()] = UNLOCKED;
    lock->turn = other_proc_id();
}
```

Алгоритм Петерсона

```
int turn;
int interested[2];

void peterson_lock(int process) {
        int other = 1 – process;
        interested[process] = TRUE;
        turn = process;
        while (turn == process && interested[other] == TRUE)
            ;
}
void peterson_unlock(int process) {
         interested[process] = FALSE;
}
```

План лекции



- Средства коммуникации
- Алгоритмы синхронизации
  - Синхронизация в централизованных ОС
  - Синхронизация в сетевых и распределенных ОС
- Проблемы при работе с ресурсами

Механизмы обеспечения синхро-низации

Аппаратная поддержка взаимоисключений

- test_and_set(a, b) (проверить и установить)
- Выполнение команды заключается в следующих двух действиях, причем выполняемых атомарно:
  - значение переменной b копируется в a
  - значение переменной b устанавливается в истину
- Практически все основные архитектуры имеют подобную команду в своем составе
  - В SPARC-архитектуре существует команда ldstub (load store unsigned byte): ldstub [addr],reg
    - В результате выполнения этой команды содержимое памяти по адресу addr копируется в регистр reg, а все биты памяти addr устанавливаются в единицу
  - В x86 архитектуре существует команда xchg
  - …

Программный интерфейс команды test_and_set

- Программный интерфейс команды test_and_set. Используется реальная ассемблерная команда ldstub

```
int test_and_set(volatile int* addr) {
    asm(ldstub [addr],reg);
    if ( reg == 0)
        return 0;
    return 1;
}
```

Механизмы обеспечения синхро-низации

Блокировка с запретом прерываний

```
enum state { UNLOCKED, LOCKED };
typedef int lock_t;
void init_lock(lock_t* lock) {
    *lock = UNLOCKED;
}
void lock(volatile lock_t* lock) {
    disable_interrupts;                    /* запрет прерываний */
    while ( lock != UNLOCKED) {
        enable_interrupts;                 /* разрешение прерываний */
        disable_interrupts;
    }
    lock = LOCKED;
    enable_interrupts;
}
void unlock(lock_t* lock) {
    disable_interrupts;
    lock = UNLOCKED;
    enable_interrupts;
}
```

Спасибо!

Вопросы?