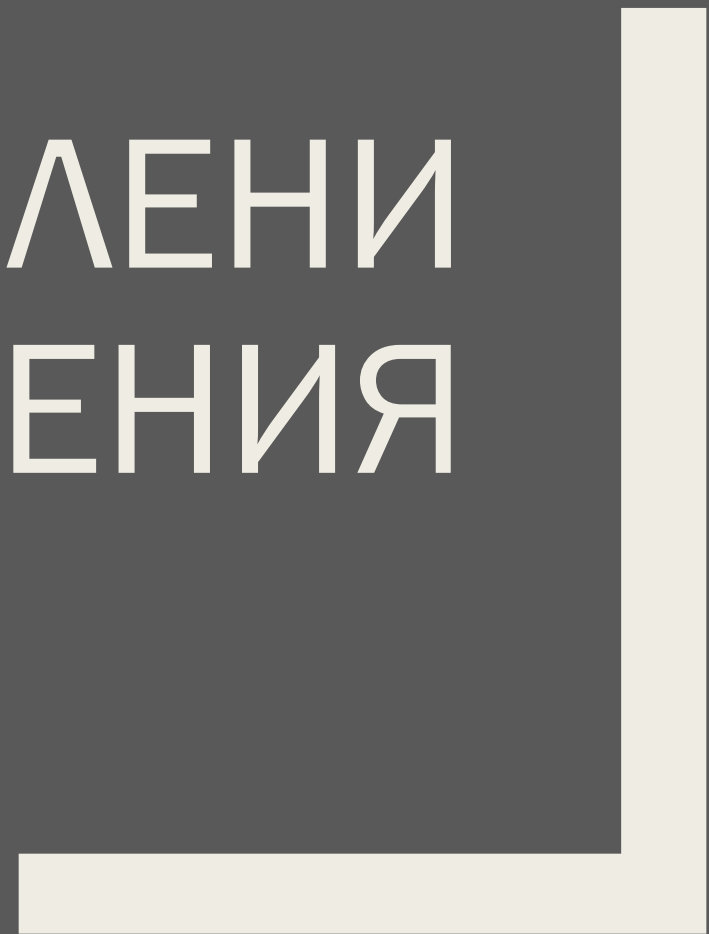


РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ

Павел Кюркчиев
Ас. към ПУ „Паисий Хилендарски“
@rkyurkchiev

РАЗПРЕДЕЛЕНИ ПРИЛОЖЕНИЯ



История на приложенията:

- Първо поколение - Централизирани изчисления (Centralized computing).
- Второ поколение - Разпределени приложения (Distributed applications).

Централизирани изчисления (Centralized computing)

- Централизираните изчисления обединяват и контролират всеки аспект на приложението, включително бизнес процесите, управлението на данните, както и графичната визуализация.

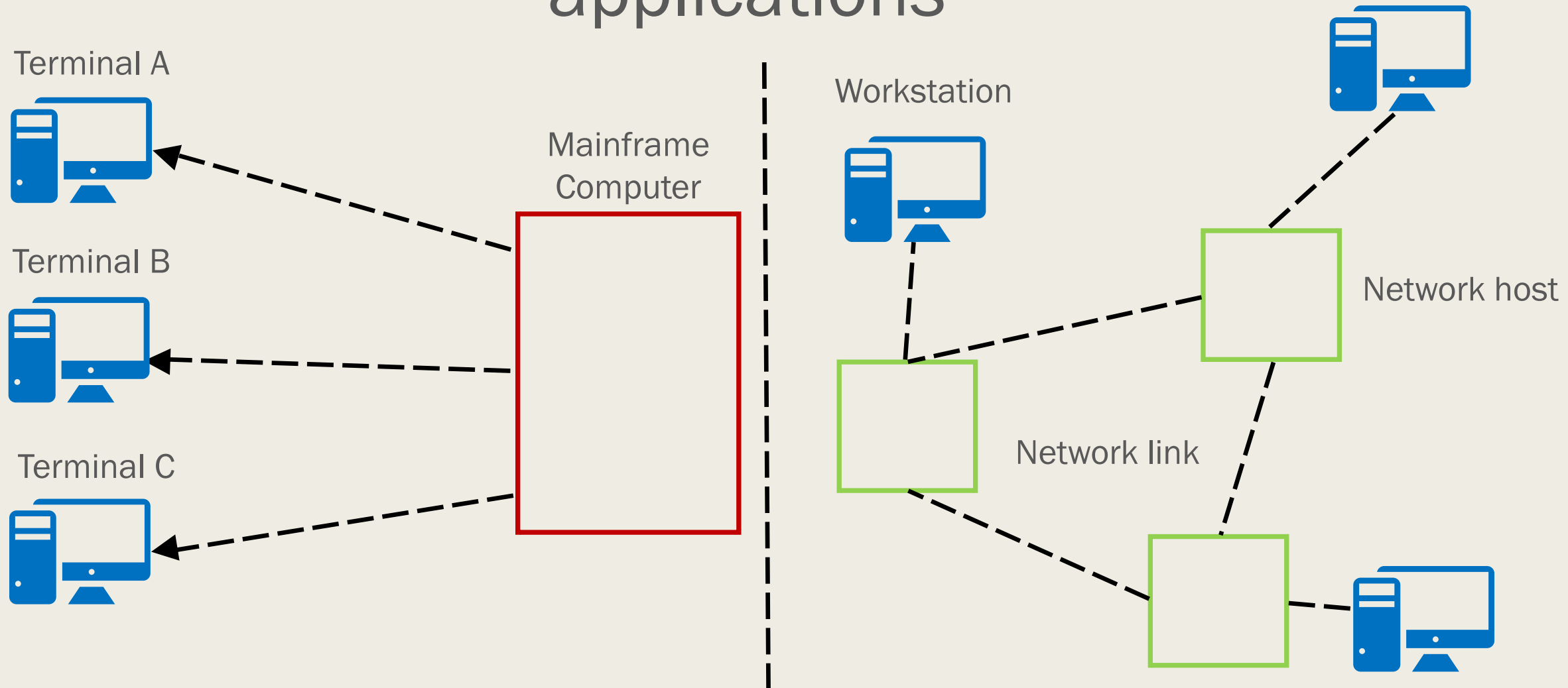
Проблеми на Централизираните изчисления (Centralized computing)

- Цялата тежест на процеса, включително достъпът до данните, бизнес логиката и презентационната логика, е представена в едно приложение на една физическа машина.
- Големите и комплексни приложения са трудни за поддръжка и развитие.
- Тяхната откъсната и комплексна логика прави интеграцията им с други приложения и платформи изключително трудна.

Разпределени приложения

- Разпределено приложение е приложение, физически разделено на отделни компоненти, чието обединение описва пълната функционалност на приложението.

Centralized computing vs Distributed applications



Разпределено програмиране

- Разпределеното програмиране се характеризира с няколко физически компонента, работещи в синхрон като една обединена система.

Физически компоненти?

- Могат да бъдат както ПРОЦЕСОРНИ ЯДРА, така и КОМПЮТРИ, НАМИРАЩИ СЕ В ЕДНА МРЕЖА.

Основата на разпределеното програмиране е оптимизацията.

- Ако даден компютър може да приключи определена задача за 5 секунди, то 5 такива машини би трябвало да приключат същата задача за 1 секунда.

Проблемът се състои в това да се съчетаят отделните модули и системи, така че да работят паралелно.

Пример



Основни принципи на разпределените приложения

- В книгата си „Distributed .NET Programming in C#“ Том Барнаби описва пет основни принципа на разпределените приложения.

5-те принципа на разпределените приложения

Distribute Sparingly

Localize Related Concerns

Use Chunky
Instead of Chatty
Interfaces

Prefer Stateless Over
Stateful Objects

Program to an
Interface, Not an
Implementation

Умерено разпределение (Distribute Sparingly)

- Този принцип се основава на базови факти, свързани с изчислителните машини. Извикването на метод от инстанциран обект в различен процес е сто пъти по-бавно от извикването на метод от същия процес. Преместването на обект от една машина на друга в една и съща мрежа и извикването на метод от него може да забави процеса на изпълнение до 10 пъти.

Кога трябва да разпределяме?

Отговор

- Само когато е необходимо.
- Най-често започваме да разделяме една система от базата данни. Доста често базата данни се намира на отделна машина, наречена сървър, с други думи, е разпределена в отделен слой.

Причини

- Базата от данни е сложен, скъп и обикновено доста тежък софтуер, нуждаещ се от силен хардуер.
- Базата данни може да съдържа релационни данни и да е споделена между много приложения. Това е възможно само ако всички приложения имат достъп до базата данни.
- Базите от данни са създадени да работят като отдалечени физически слоеве.

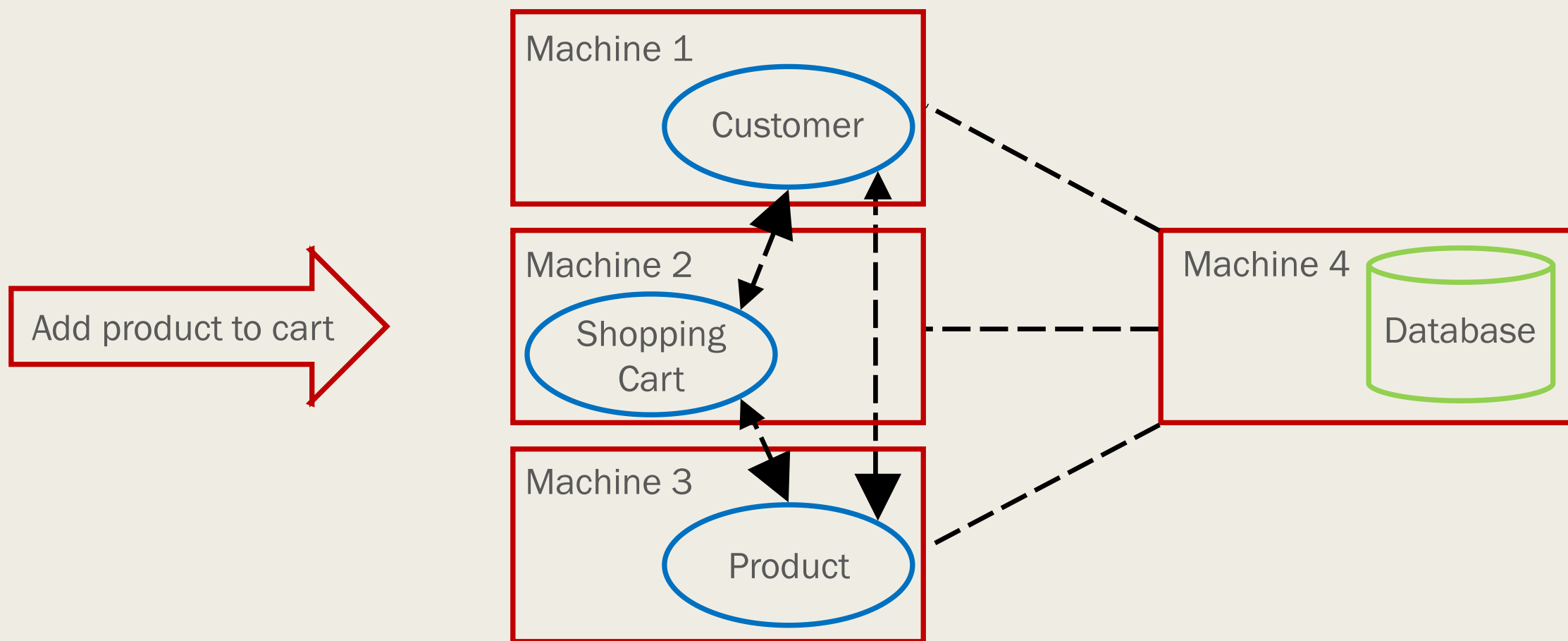
Пример

- Ако разгледаме като пример презентационната логика, тук нещата изглеждат малко по-сложни. Най-важното нещо е да знаем пълната спецификация на приложението. Ако на потребителите им се налага да достъпват отдалечен сървър или терминал (като например АТМ на някоя банка), може би е добра идея част от логиката да бъде изнесена там.

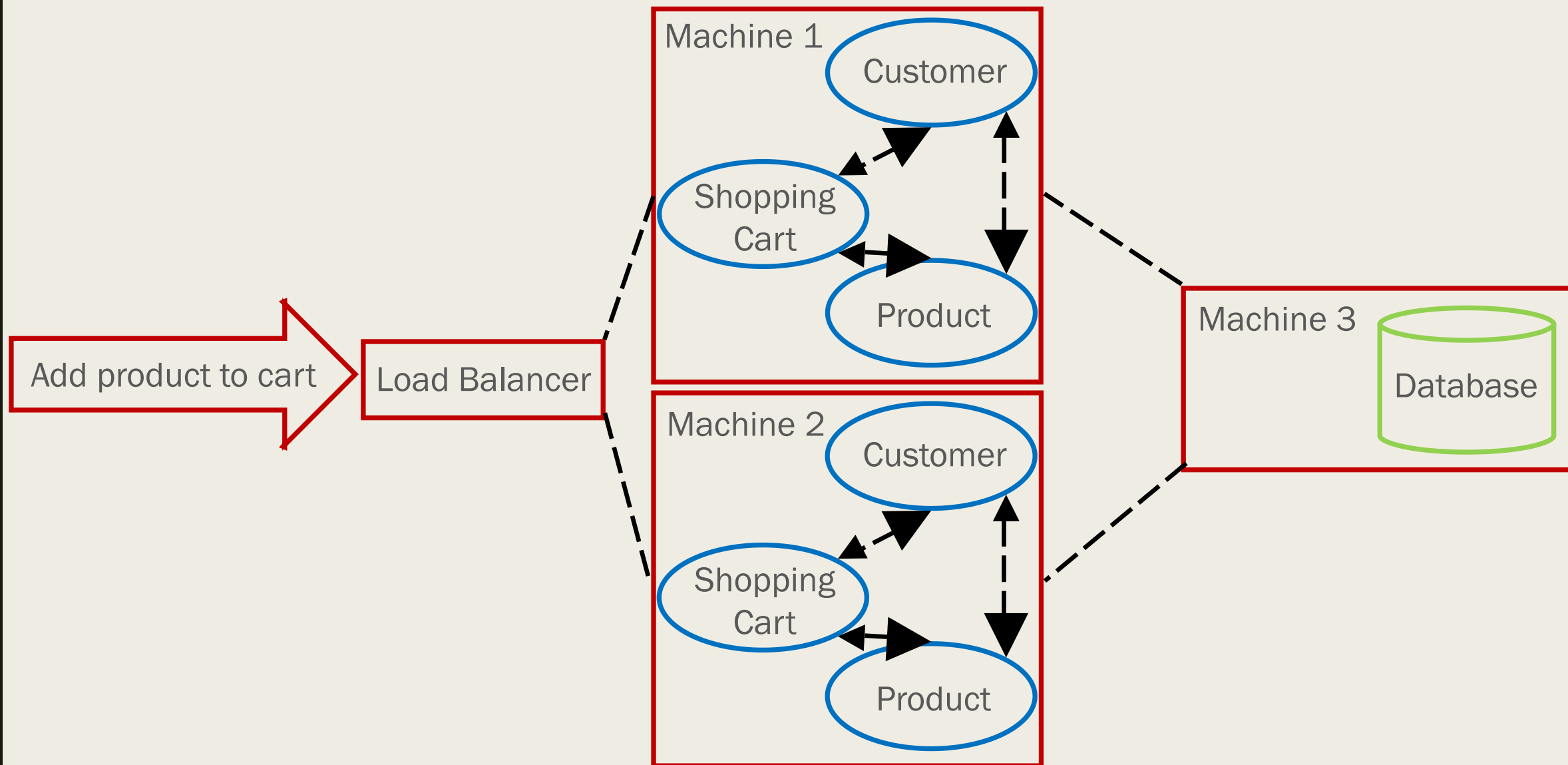
Локализиране на свързващите отношения (Localize Related Concerns)

- Ако бизнес логиката на едно приложение бъде разделена на отделни машини, то много внимателно трябва да се подсигури комуникацията между най-често общуващите компоненти. С други думи трябва да се локализират свързаните модули.

Паралелно изпълнение на отделни процеси от бизнес логика



Разпределение на натоварването



Малък интерфейс вместо комуникативен (Use Chunky Instead of Chatty Interfaces)

- В основата на този принцип е заложена идеята функционалността да е максимално близо до потребителя, който ще я използва.

Пример:

- Комуникация между два сървъра, намиращи се на големи разстояния един от друг.
- На сървър 1 се намира имплементация на обекта Customer, а сървър 2 се опитва да го достъпи.


```
class Customer
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    // etc. for Street, State, City, Zip, Phone ...

    public void Create();
    public void Save();
}
```

Chatty interface.

```
class Customer
{
    public void Create(string FirstName, string LastName, string
Email, /* etc for Street, State, City, Zip, Phone ... */);

    public void Save(string FirstName, string LastName, string
Email, /* etc for Street, State, City, Zip, Phone ... */);
}
```

Chunky interfaces

```
[Serializable]
class CustomerData
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    // etc for Street, State, City, Zip, Phone ...
}
class Customer
{
    public void Create(CustomerData data);
    public void Save(CustomerData data);
}
```

Оптимизация на Chunky interfaces

Prefer Stateless Over Stateful Objects

- Препоръчителен и не задължителен.
„Stateless“ обектите са за предпочитане пред „Stateful“ обектите.

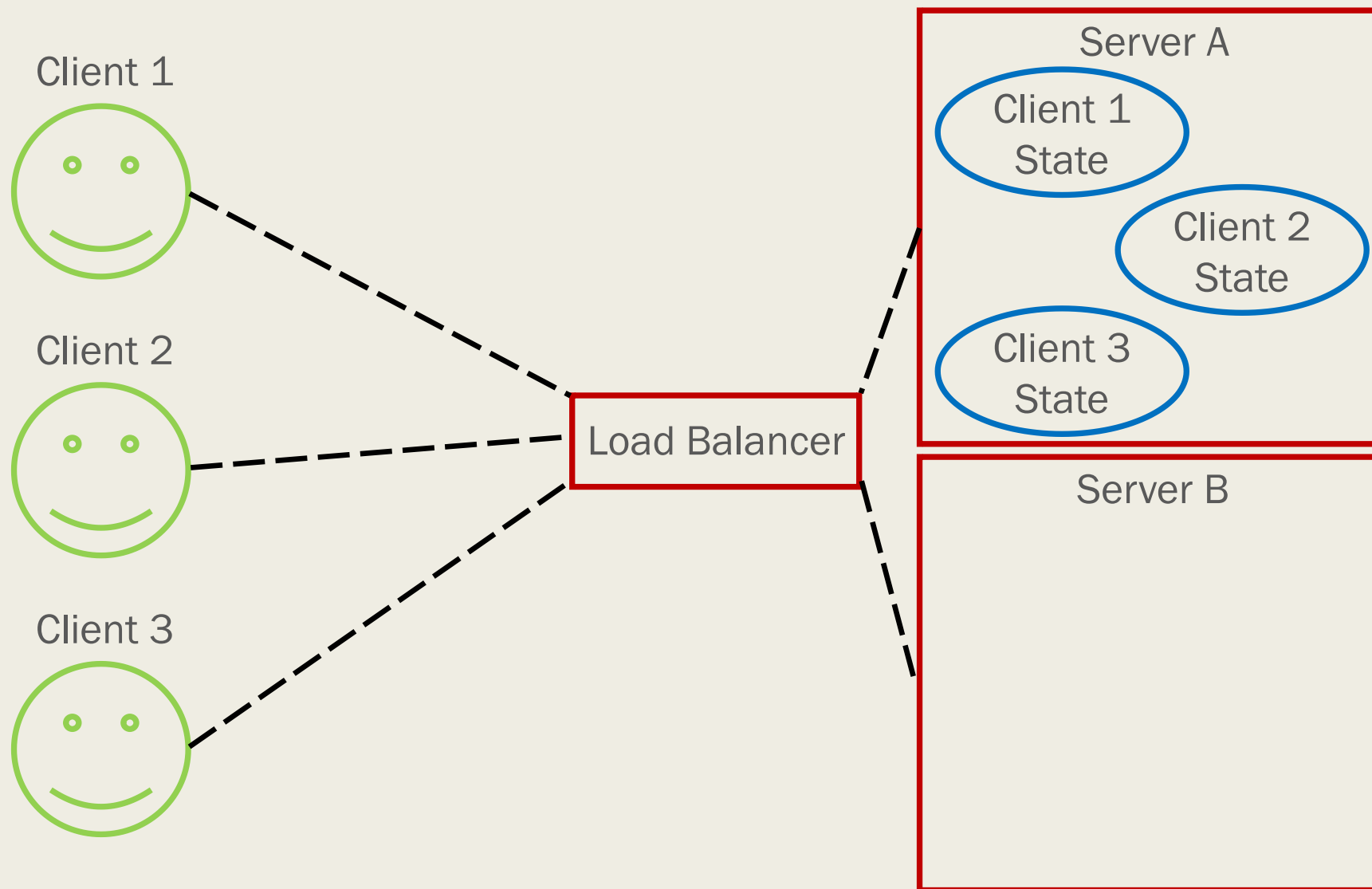
Какво е “Stateless”?

- Това са обекти, които могат безопасно да бъдат създавани и унищожавани между различни методи и процедури. Не е задължително даден метод да унищожи инстанцията на обект след като бъде изпълнен, за да бъде "stateless". Но ако това се случи, премахването на обекта не трябва да окаже промяна в поведението на приложението.

Проблеми на “Stateful” обектите

- Тези обекти съществуват на сървъра за удължен период от време. В този период от време може да се буферира голямо количество системни ресурси. Биха могли дори да предотвратят заделянето на ресурси за други обекти.
- Другата негативна черта е намаляването на ефективността на дублирането и load balancing на приложенията, разделени между много сървъри.

Схема за идентификация на



Program to an Interface, Not an Implementation

- В основата на разпределеното програмиране стои интерфейс базираното програмиране. Проблемът, който се решава с този подход, е повече свързан с по-лесното прехвърляне на проект върху отдалечен сървър, а не с неговата бързина или разширяемост.

ВЪПРОСИ ?

