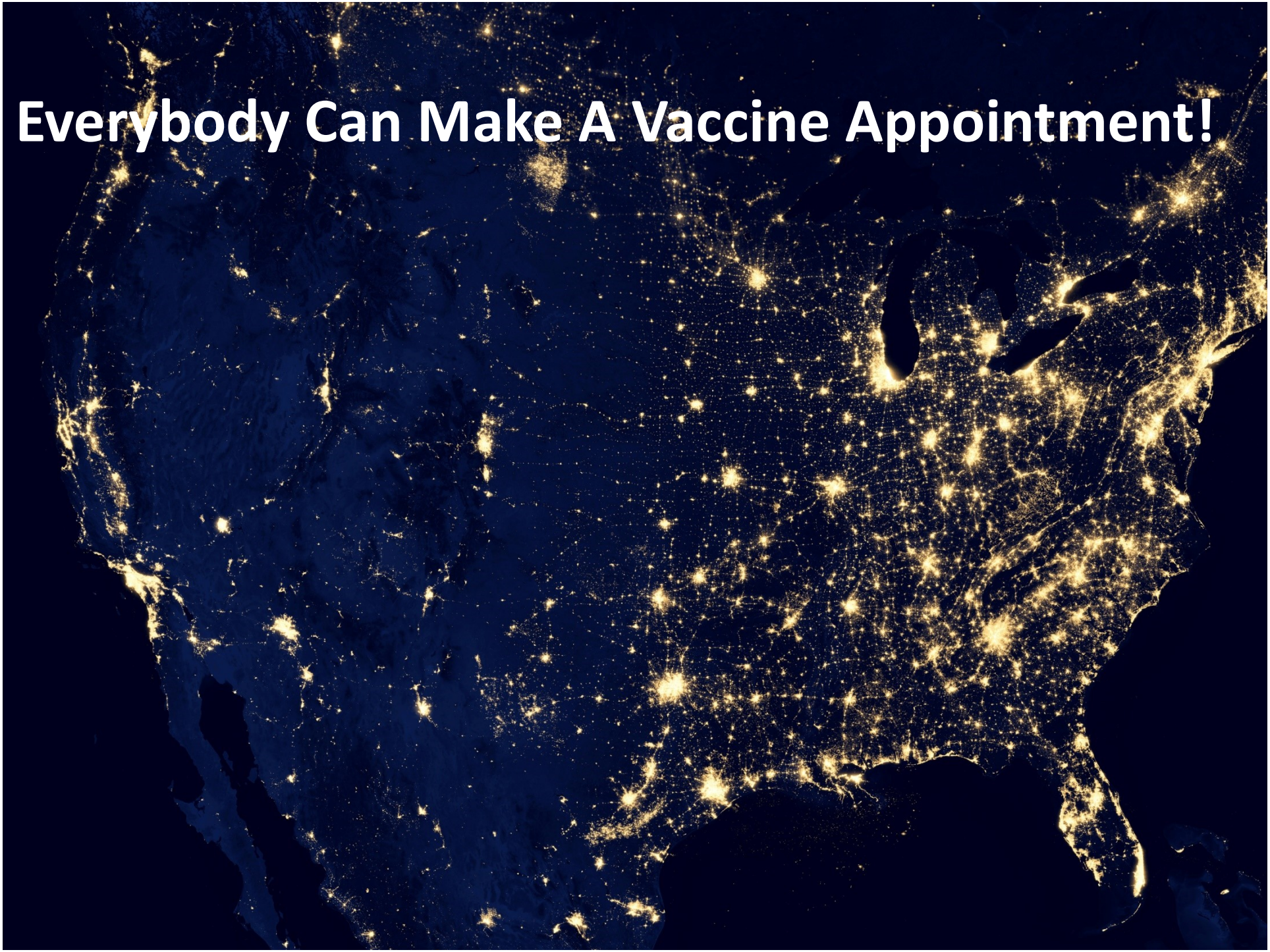
# Architecting a Vaccine Website is Harder Than You Think

Michael Stiefel
Reliable Software, Inc.
development@reliablesoftware.com

# We Finally Have a Coronavirus Vaccine!


Everybody Can Make A Vaccine Appointment!

Our Problems Are Over!


No!
WRONG WAY

# The User Experience Was Initially Horrible

User Interface Often Crashed

User Had to Repeatedly Enter Information

If You Did Not Get Appointment – Start Over!

Many Entry Points – Difficult To Navigate

Why?

Could we architect a better solution?

# What does an architect do?

The architect is responsible for the conceptual integrity of the user experience

# User Experience Includes Emergent Application Properties

User Interface

Performance

Scalability – Throughput, Geography, Users

Cost

Security

Reliability

Ease of Deployment

Workflow

Emergent Properties are a Constraint on Every Use Case or User Story

The architect mediates between the business people and the technologists.

Architecture is not about software design or craftsmanship.

An architect might have coding responsibilities in addition to architectural responsibilities.

# What then is the Architectural Problem?

# Requirements

Vaccine Availability

Vendors report vaccine availability

Efficiency vs. Priority – externally determined

Eligibility Not Validated by the System

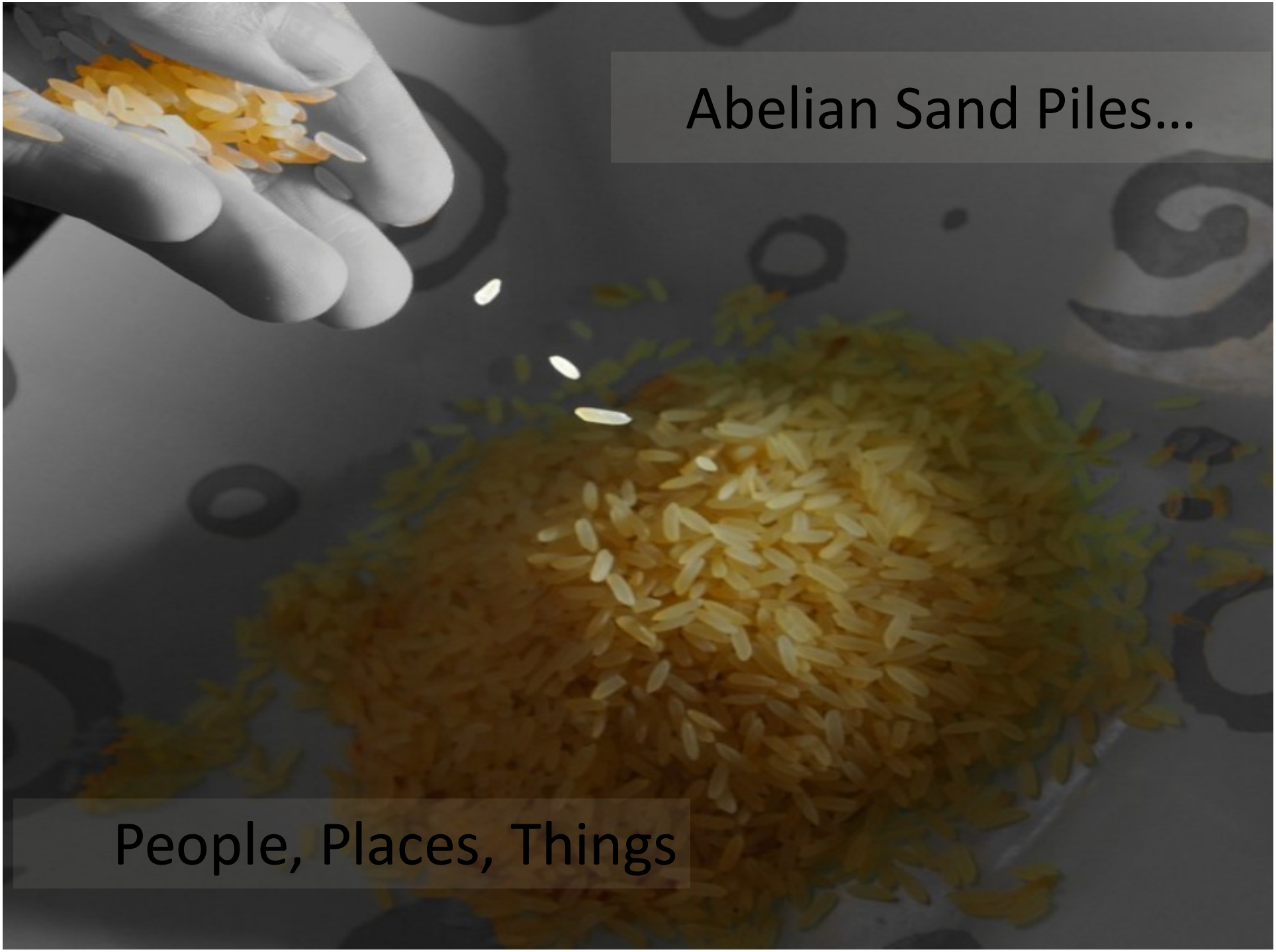
Finding Appointments

Filling out Required Information

Schedule Appointments (1 or 2)

Administrators can track progress, diagnose system

# Critical software issue: Resource Contention Under Constraints


Abelian Sand Piles…
People, Places, Things


# The Wall Street Journal Effect

When you can no longer vertically scale –
it is too late.

# Real World Resources

Books

Vaccines

Concert Tickets

Airline Reservations

# Computing Resources

Compute
Storage
TCP/IP Connections
Buffers
Queues
Database Connections
Database Locks
Third Party Services and Libraries
The Internet/World Wide Web

# Our System Constraints

# Allocation of Vaccines

Current Constraint is lack of supply.

Feds to State

Feds to National Pharmacy Chains

# Priority Order

Some Vaccines Require Reservation for Second Shot

# User Should Get a Reasonable Response

# What does such an Architecture Look Like?

Must Program for Supply < Demand

# Must Design For Failure

When the failures start…it is too late.

The more dependencies, the more opportunities for failure, the greater the consequences

Avoid single points of failure.

No single point of failure => a distributed system

Software State ≠ State of the World

# User Interface/Ease of Use/Reasonable Workflow/Driven by Software Services

# Seven Principles of Scalable Architecture

The crucial issue is how to respond to failure. The underlying infrastructure cannot guarantee availability.

Failures Cascade - an unhanded failure in one part of the system becomes a failure of your application.

There is no such thing as a transient failure. Fail fast and treat it as a resource failure.

Use a Margin of Safety when determining how many resources you need

Eliminate single points of failure. Accept that you have to build a distributed application

Degrade gracefully and predictably

Assume the Rare will Occur because It Will

# How Would We Try to Solve the Architectural Problem?

# Where does are application need to scale? Where are the points of failure?

People

Vaccine Availability

Appointment Slots

Examples of how others have attempted to solve this problem

Amazon

Ticketmaster

# Airlines

Healthcare.gov

# Why these approaches will not work here

Scarce Supply – No Optimistic Concurrency

Vendors unwilling to use Central System

Lack of Control Over Supply

# Satisfying the Constraints

# Portal Based Application

Make clear to the user the threefold opportunities and scheduling difficulties

Direct access to Mass sites through pre-registration

Automated data entry of other web sites

Updated, cached supply information

No time to develop web services

Use cloud for scalability

You cannot afford to buy resources for peak demand.

Use relational databases only when necessary

Use cached information for scheduling availability

# Separate data stores to minimize lock conflicts

User Information

- Allow Pre-registration

Scheduling Locations

Vaccine Availability

Available Appointments

No business logic in the database

# Separate out services by access path

Enter User Information

Query for Appointments

Schedule Appointments

# Fail Fast, Aggressive Timeouts

Report back to user as quickly as possible what happened, do not just crash

Can degrade to lower quality service

Prioritize scheduling appointments over entering information

Prioritize sending to a different website over entering information

Disability Friendly

Mobile friendly for younger people

# Conclusions

Political, social, economic constraints are real

You have to architect for the conditions you have, not the conditions you would like to have

You cannot scale after the fact

You have to architect for failure

Understand the scale boundaries in your application

How would you architect the system?