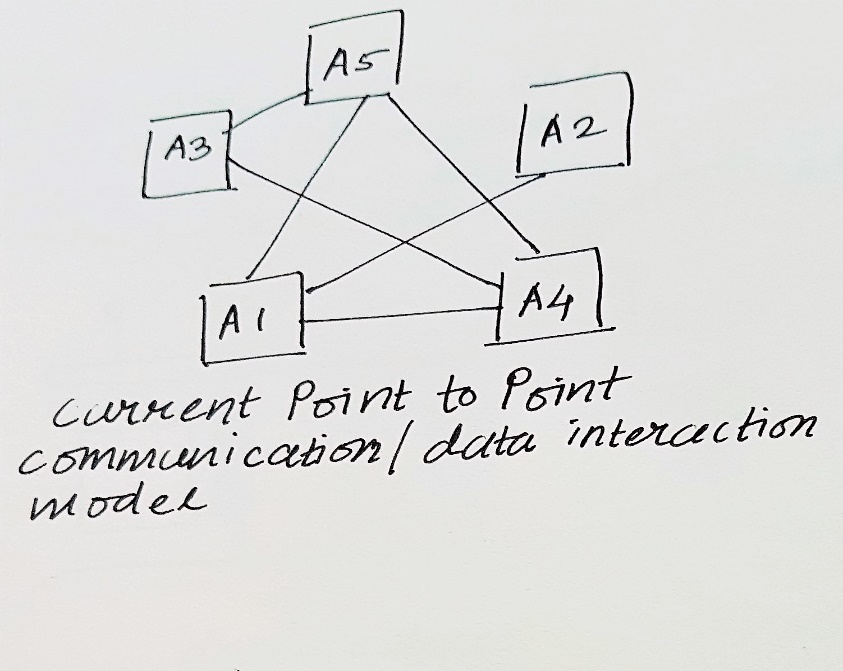
Data Loader Ideation.

Current Scenario:

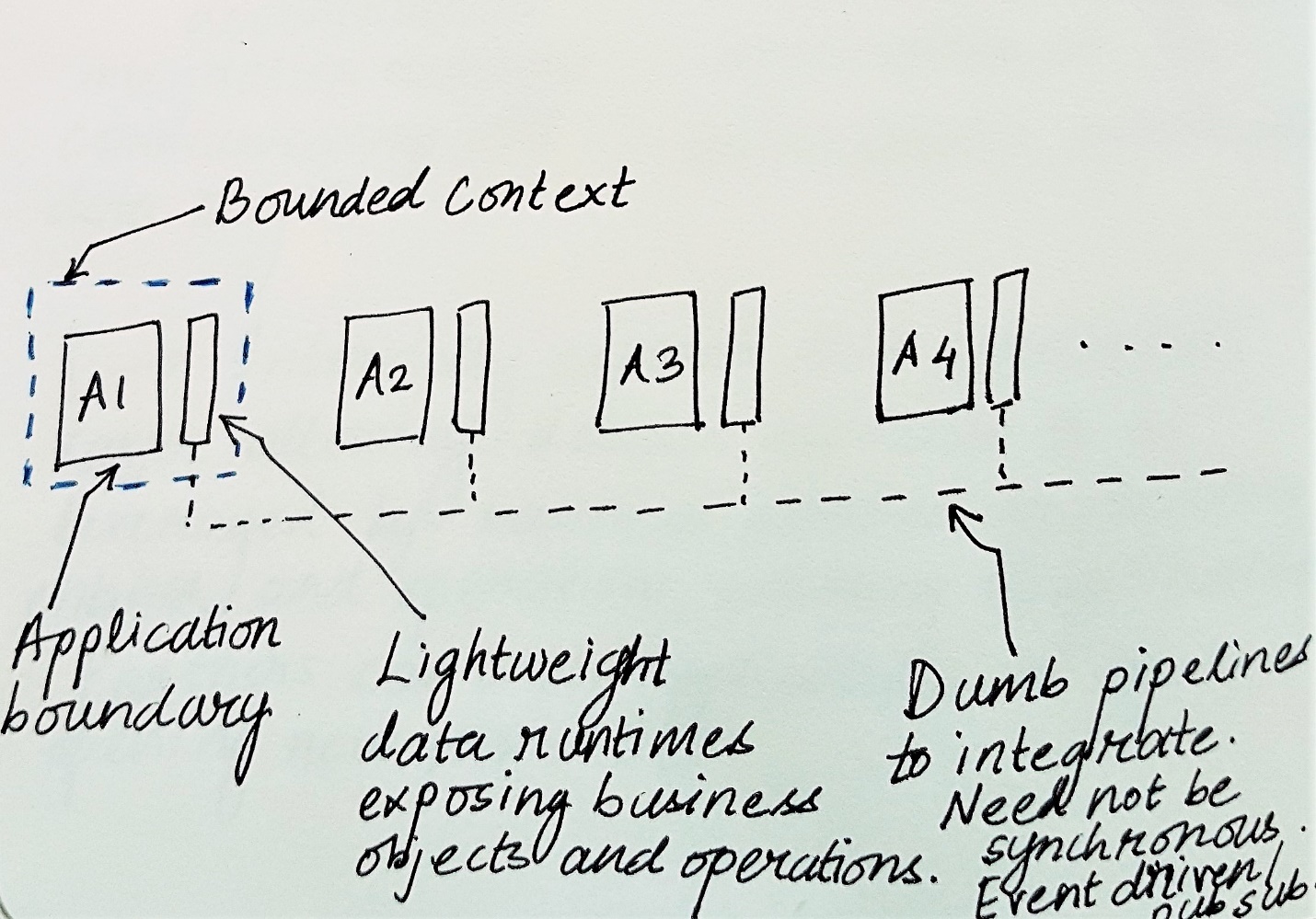
* Each application encapsulates and solves a domain problem with domain expertise built into it.
* Each application uses its own system of records to persist this domain information, tenant configuration, application configuration and data in a way best suited for it.
* Practically, an enterprise application cannot have a clear bounded context of domain and hence delegates to other application(s) to leverage their functionalities within it, providing client with aggregate features.
* Business evolution has resulted in applications coordinating in a point-to-point fashion with few mixed models like pub-sub or data replication depending on their requirements.
* Models of domain and their operations are deeply embedded within each application making it difficult for services to manipulate these domain objects without being a part of it.
* Some high-level external APIs are exposed enabling bulk uploads, external integrations, service requests/calls etc.

Problems Faced:

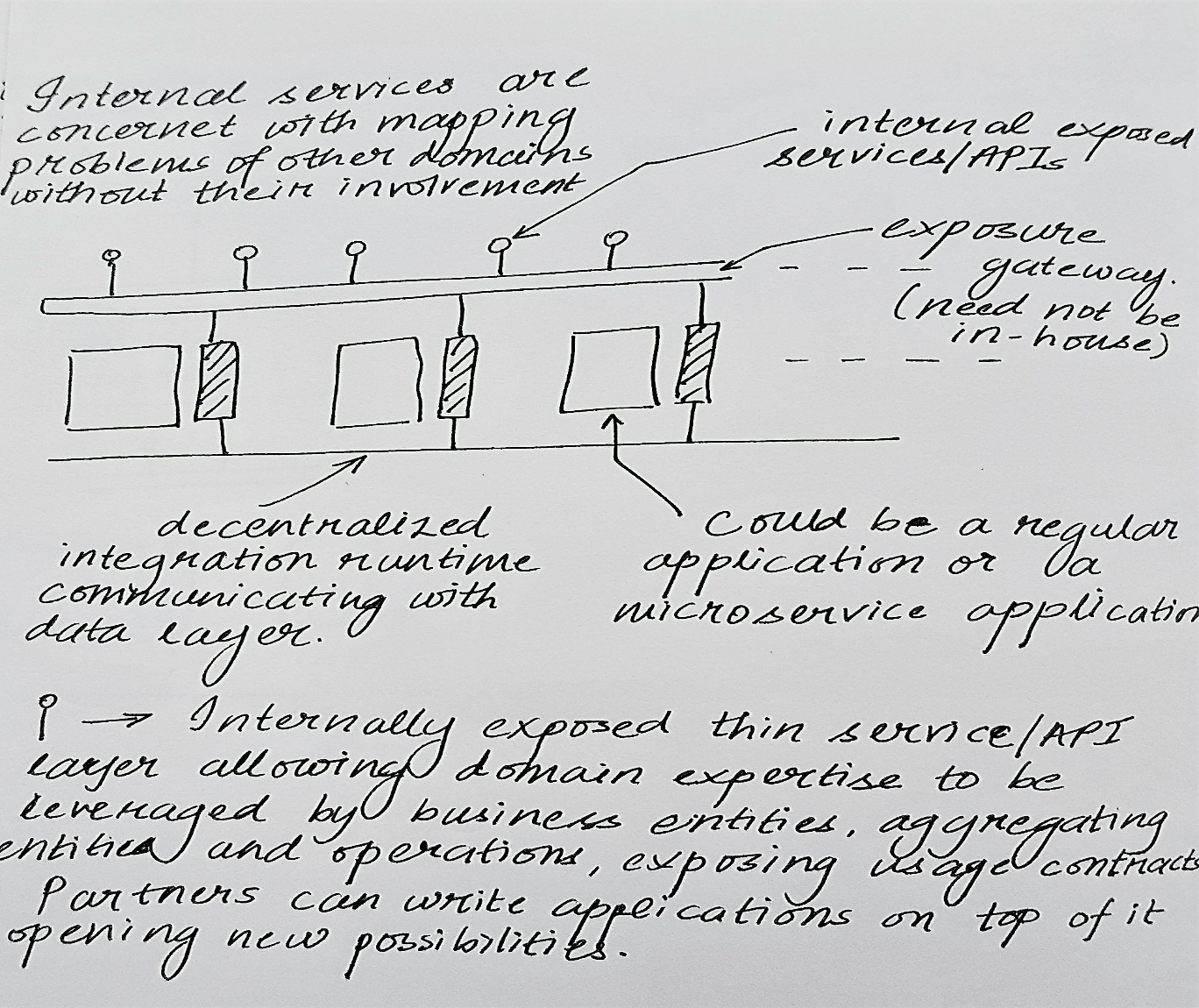
* Tenant migration and data migration between setups and environments is difficult, error ridden and time consuming. Analysts spend a lot of time solving these problems.
* Cannot migrate a tenant from one environment to another across complete Zycus suite in one click.
* Slow creation of completely configured QA setups and test bed creation.
* Scenario based automations that are slow to develop ( because of time needed to understand product’s system of records ), fragile, not standardized and non-reusable. The time spent on product’s KT is not reused (requirements change, people change) resulting in huge wastage of time.
* Involvement of multiple teams (Development team, L2, DBA, RM) with a slow process in executing CSCRs.
* No tenant configuration/data versioning (Environment x Product x Tenant) and pipeline line like promotion for deploying usable products with client data already configured.
* Missing the ability to stage a product’s replica configured with production data for analyzing/debugging an issue on production and hot fixing.
* Partner projects like RPA, integration, infrastructure abstraction, analytics are slow and cannot be realized to full potential because there is no standard way of communicating with apps and their data models.

Proposal:

* Develop light weight, decentralized, data-integration runtimes (data-apps) that will leverage code already written in apps for data access and manipulation. These supplementary applications can be developed in parallel by dedicated team without affecting the main application’s roadmap and will model the domain so that domain expertise can be leveraged by all teams not directly involved in product development and problems of other domains/teams can be interfaced and solved without requiring breaking changes in main applications.



* Instead of developing components within applications requiring stop-the-world changes, they can be developed in parallel and integrated with apps in future (if desired) using train-deployments and free business application developers to focus on business features.
* These data-apps can be internally integrated with each other using any suitable model like centralized ESBs, distributed pipelines or pub-sub models. The integration can be easily evolved as end points are smart and pipeline is dumb.
* Loosely coupled API abstraction layer or gateway can be built on top of these data-apps for having a logically centralized layer for exposing domain models and operations for internal uses.



* As these models mature, connectors for them will be developed. This internal exposure gateway will enable for development of partner apps (like RPA and analytics) and engagement apps (like external synchronous integration, improving online persona) and improve reusability.
* Finally, there can be separate service layer(s)/API gateway(s) for supporting different granularity of operations based on front-ends/consumers like web, mobile, programmatic clients thereby having a backend for front-end model (BFF) and implementing gateway features like encryption/decryption, throttling, authentication etc. can be separated.

