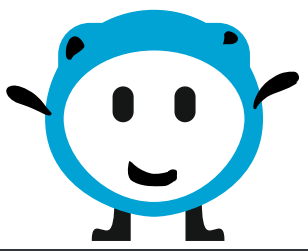


ANATOMY OF A BOT

1: COGNITION AWARENESS (CONNECTIONS)

MEMORY FUNCTIONS

TEMPORAL	CONTEXTUAL	RELATION / CUSTOM
A core memory function based on subliminal approach. Automatically records timestamps of all its interactions. Builds priority channels from its reference point based on frequency of interaction & other relevancy factors. Optimizes its memory graphs and models from Relevancy Factors including prioritizing and forgetting. Temporal memory models exhibit distinctive traits known as Temporal Awareness of a bot. Functionally, temporal memory does not rely on any other types of memory functions. Manually adjust or train relevancy factors through CLI Also Analyze, Optimize, Find, Train, Query, Cross-ref etc. Further override or customize optimization function with 3rd party skill bots integration. Contributes to relevancy factor among bots.	Secondary memory function of a bot, supervised or automatic operation. Uses temporal awareness and other relevancy factors to generate contexts of interactions among Bots. Add or modify contexts manually and train functions. Optional 3rd party integration of skill bots to customize/optimize contextual memory function. Manually add or modify contexts through CLI Also Analyze, Optimize, Find, Train, Query, Cross-ref etc. High to medium priory contributor as default in bots relevancy factor radar.	Relational or other custom supervised memory functions can be added. Apart from default relations Parent, Child" Sibling, any type of custom relation can be added. "Kind of", "is a", "friend of" You can train custom relation and semantics through simulations, established examples and scenarios Customer "is a" Person or Company Apple "is a" fruit or apple "is like" mango. 3rd Party bots can be integrated to extend logics Ads for business "is like" key for lock (keylock relation has already been established) Relations can be converted to contexts. Highest priory contributor as default in bots relevancy factor radar.

2: CORE FUNCTION (ENGINE)

LOGICS visual blueprint ide	SCRIPTS
Data flow (Models) Function flow (Bots Networks) Interactions Restrictions And other Higher level logics	Meta Scripts Libraries Classes And other Lower level code

3: INTERFACES

INBUILT

REMOTE

INPUT	OUTPUT	APIs/Query
Any type of fields Text, Number, Date Forms/fields Commands Objects IMAGE Memory/Awareness	CLI/Panels/Views Reports/Charts/Graphs Memory/Awaness Objects Pictures/Motions/Simulations or any custom Representations	MetaAPI MemoryQL (Global/Local/Mixed) GraphQL CLI

4: IDENTITY

STATE	VERSION	TYPE
Global Local Mixed Active Limited	License/Ownership Name/Alias/HASH Description/Help Forks/Revisions	DATABOT METABOT

METABOT
firstname

BOT Structure

AWARENESS

TEMPORAL

timestamp timestamp timestamp timestamp timestamp timestamp timestamp timestamp

CONTEXTUAL

#context #context #context #context #context #context #context #context

RELATIONAL (OR CUSTOM)

:has is a child parent sibling child like

INTERFACE

INPUT

Form/Fields Objects Memories

OUTPUT

Views Charts/Graphs Reports Pictures Motion Simulation

APIs

MetaAPI GraphQL MemoryQL

CORE

LOGICS (Higher Level)

Flow
(function/data)

Interactions
Restrictions

SCRIPTS

Meta
Scripts

Libs
Classes

IDENTITY

STATE

GLOBAL LOCAL MIXED

VERSION

License ID HASH/
Name/Alias ? Help

TYPE

Databot Metabot

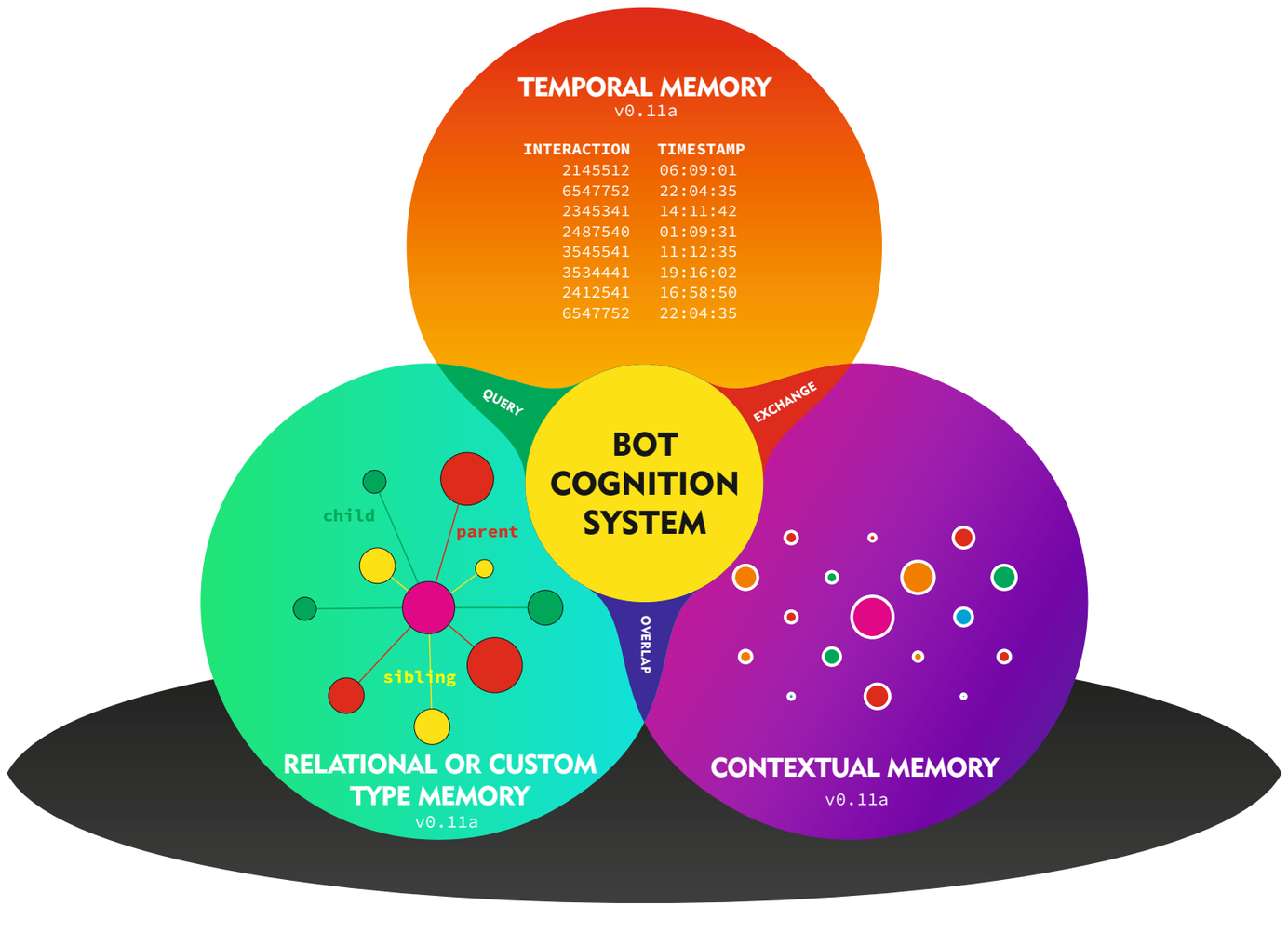
METABOTS

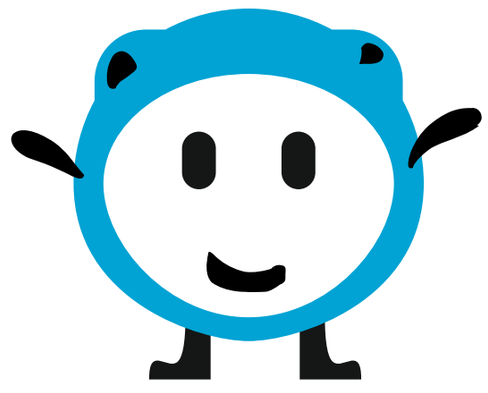
DATABOTS

DB DB DB DB

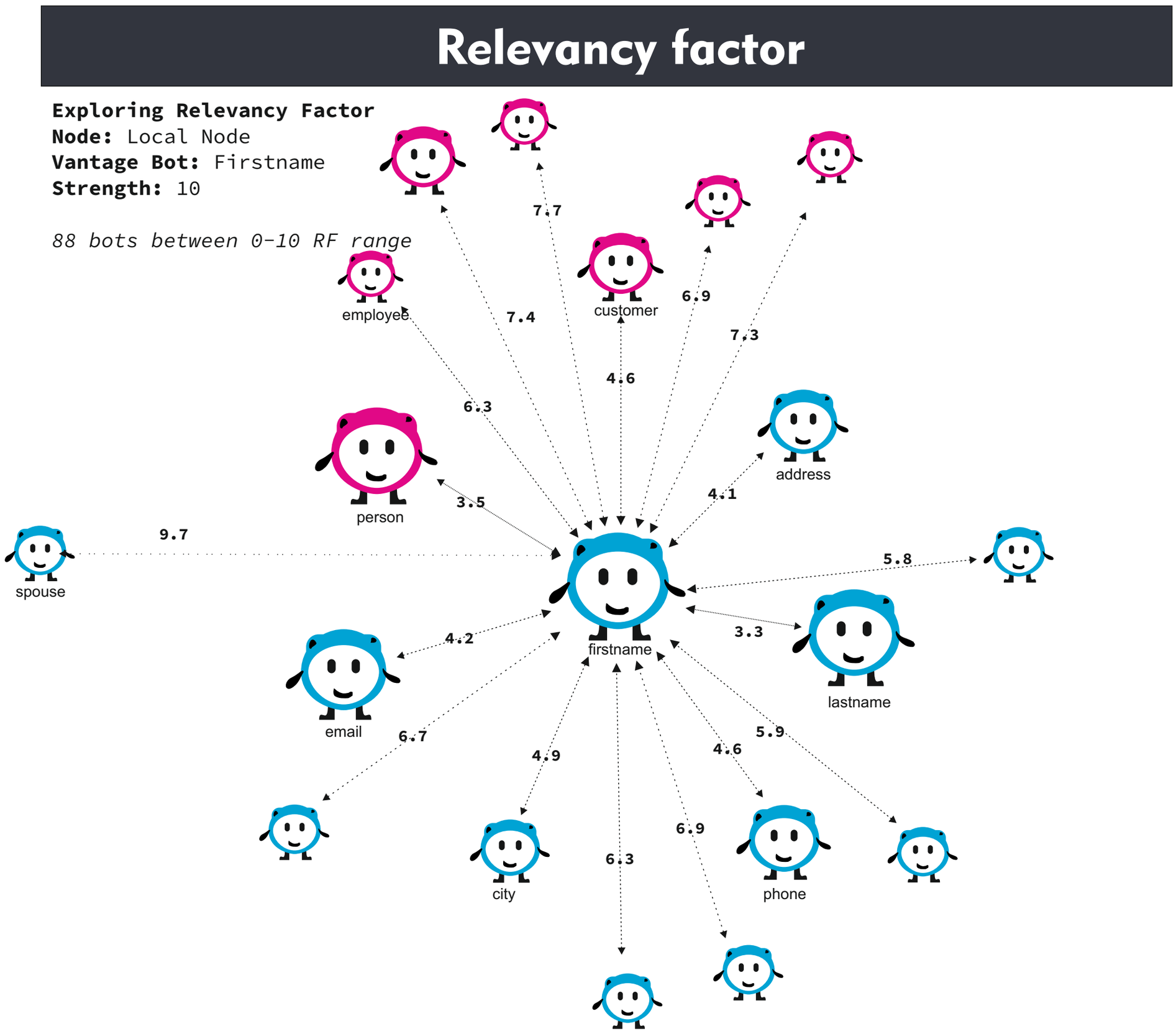
Database Layer

db layer is independent of Bots.
Metabots DO NOT have access to DB layer directly, only Databots can send data to database layer.
Database layer DOES NOT store any characteristics/features of bots.
They only store data which flow through databots.

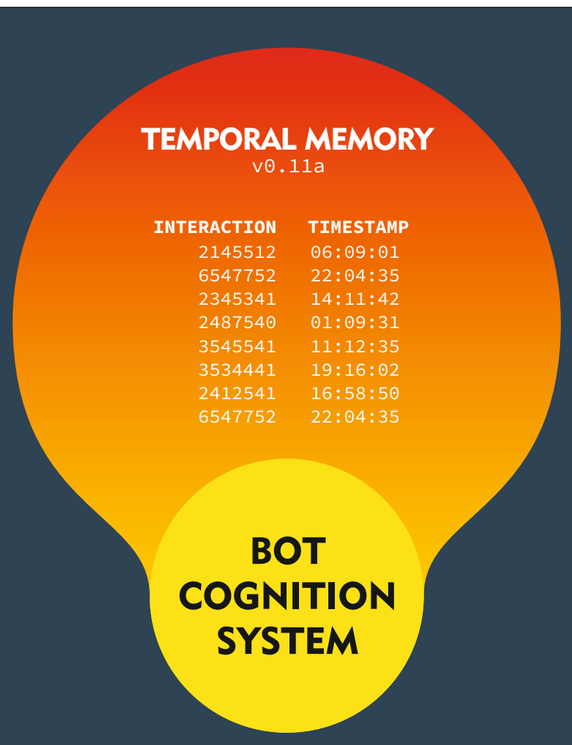




Relevancy Factor of each bot is calculated from bot's vantage point.
It uses all 3 types of memory functions to PROPAGATE relevancy value.
Manual memory functions like relation/custom are highest priority,
supervised contexts are mid priority
and temporal is the low priority contributor in relevancy factors.

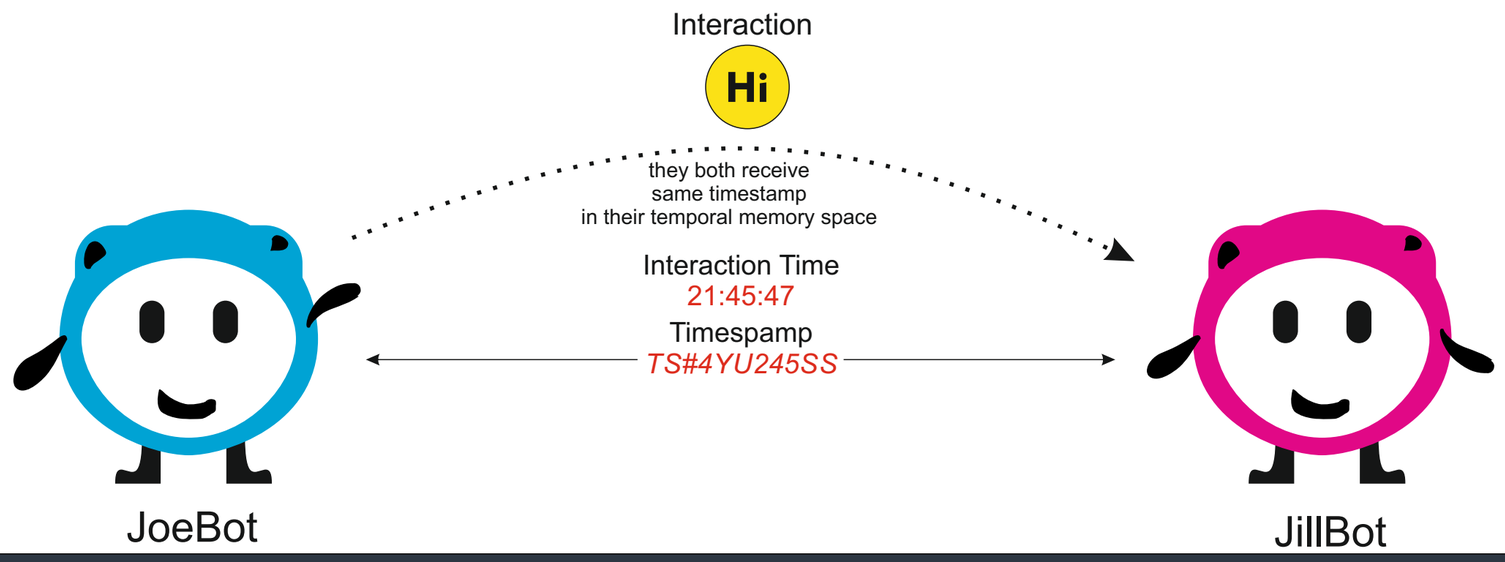


This relevancy graph is represented on local node.
Relevancy factor Bots on remote nodes will be much higher
depending on network it could be 65 to 1000.

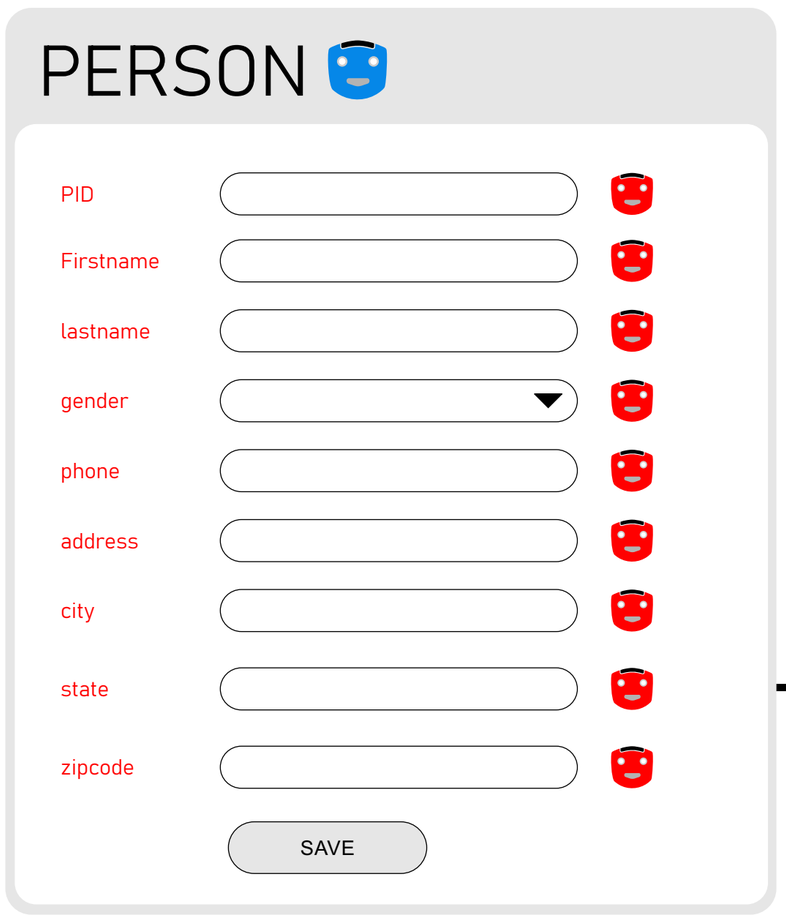


How does it work?

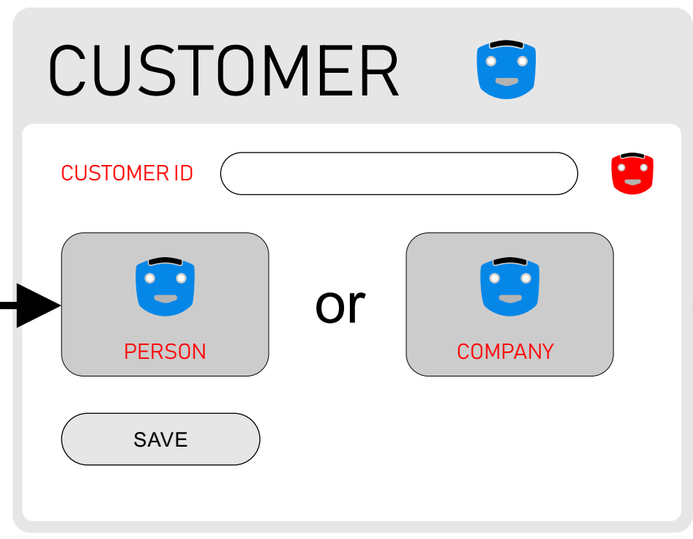
Temporal Memory function is always recording all the interactions
Timestamps are stored in Temporal memory space of bots

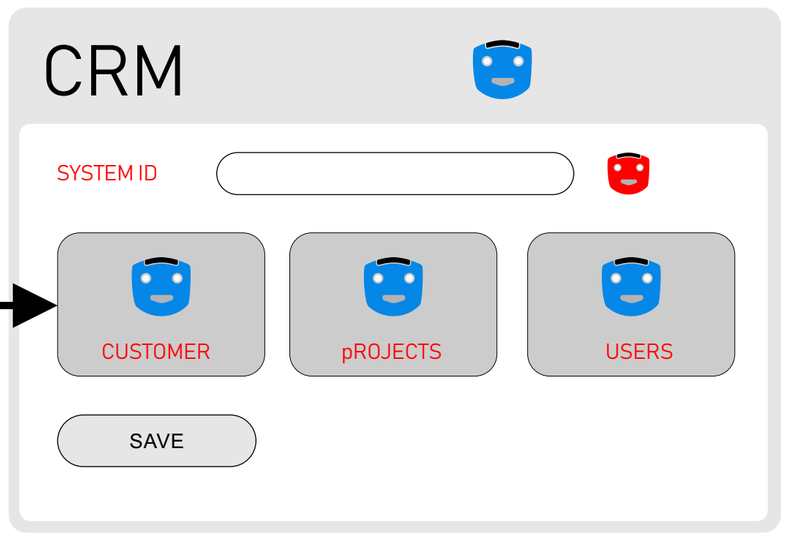


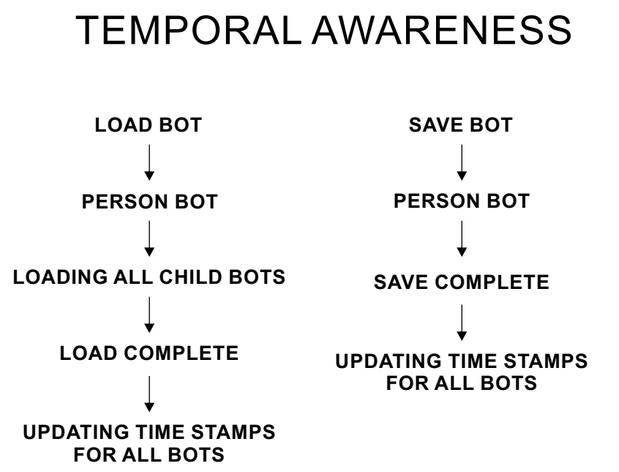
Example Case

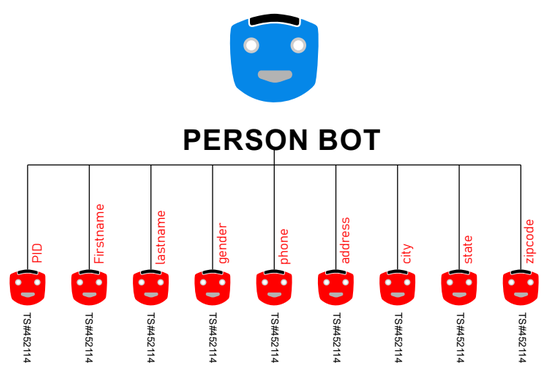


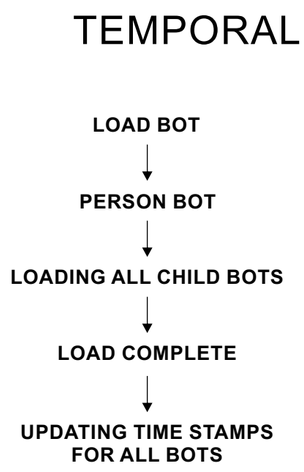
Temporal Interaction among Bots
Metabots: Person, Customer, CRM
Databots: PID, Firstname, lastname, Gender...Zipcode, CustomerID, SystemID etc...

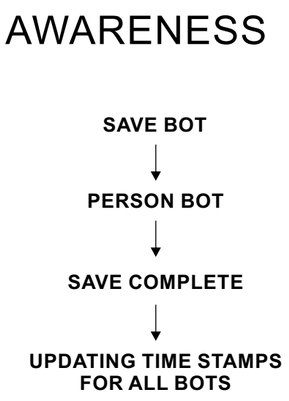


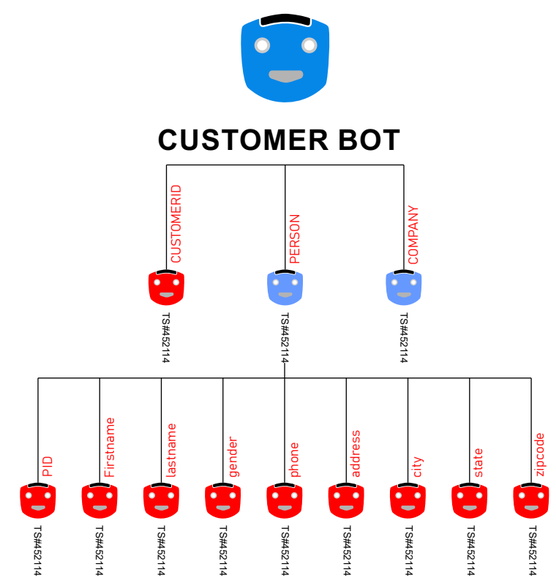










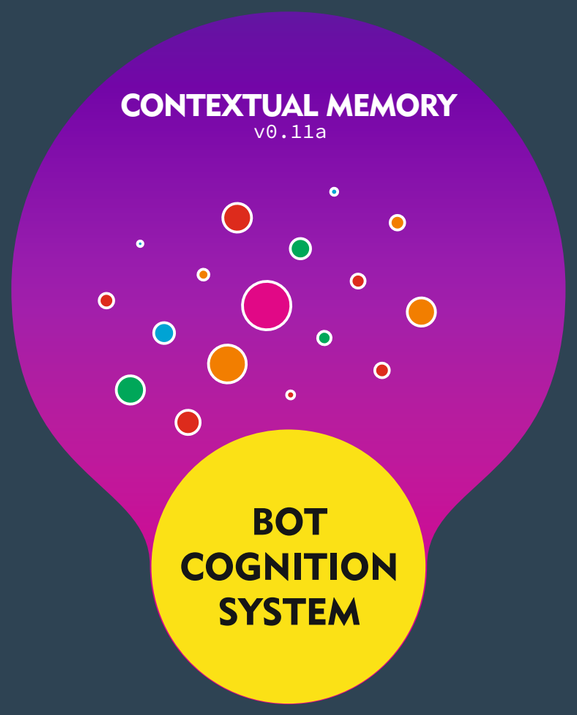


how does it contribute in Relevancy Factor?

INTERACTION	TIMESTAMP
2145512	06:09:01
6547752	22:04:35
2345341	14:11:42
2487540	01:09:31
2412541	16:58:50
3545541	11:12:35
3534441	19:16:02
6547752	22:04:35

INTERACTION	TIMESTAMP
2345341	14:11:42
5245411	12:14:02
5645542	16:47:55
3545354	11:03:27
2452424	15:37:20
3534441	19:16:02
2412541	16:58:50
3231212	14:26:45

Relevancy Factor partially depends on the Interaction Frequency
Firstname is used a lot of with Lastname so they have much stronger relevancy factors



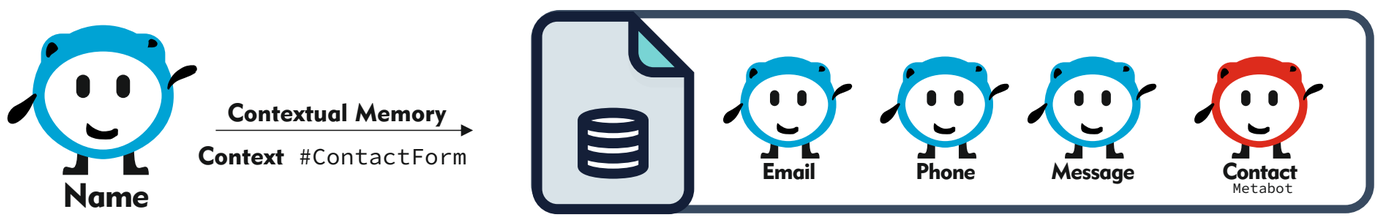
How does it work?

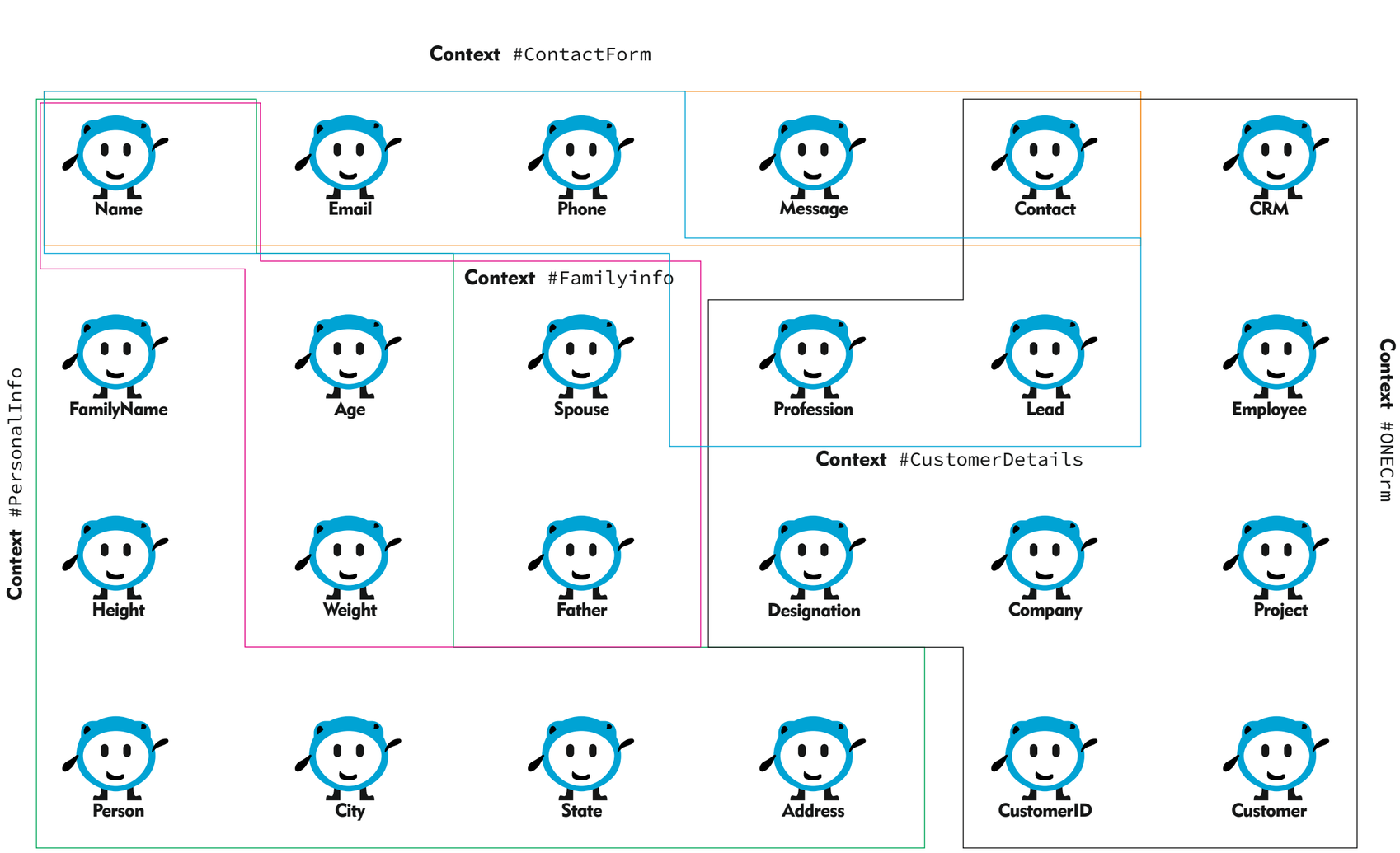
Contextual Memory function auto-generates context graphs among bots. Context itself is not a bot.
They are like a graph network (microDB) that stores reference bots group assigned to some function

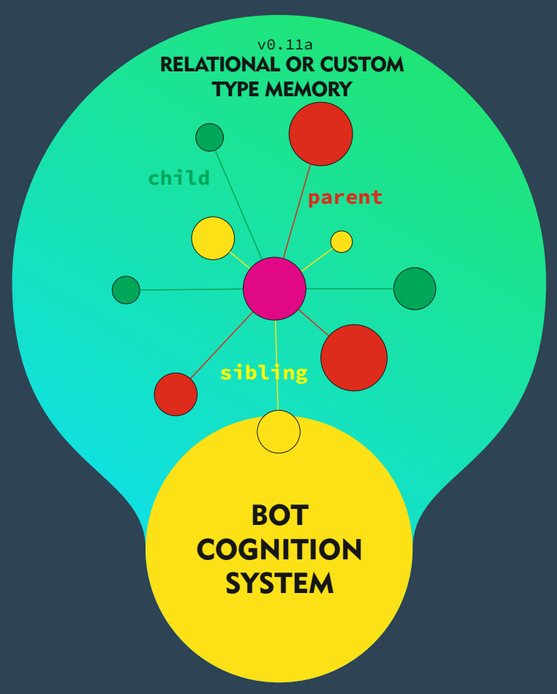
contexts	**contexts**	**contexts**	**contexts**	**contexts**
#person	#customer	#customer	#contact	#customer
#customer	#contact	#contact	#company	#contact
#employee	#employee	#employee	#CRM	#company
#contact	#company	#Guest	...	#lead
#CRM	#CRM	...		#project
#SSN	...			...
...				

Example Case

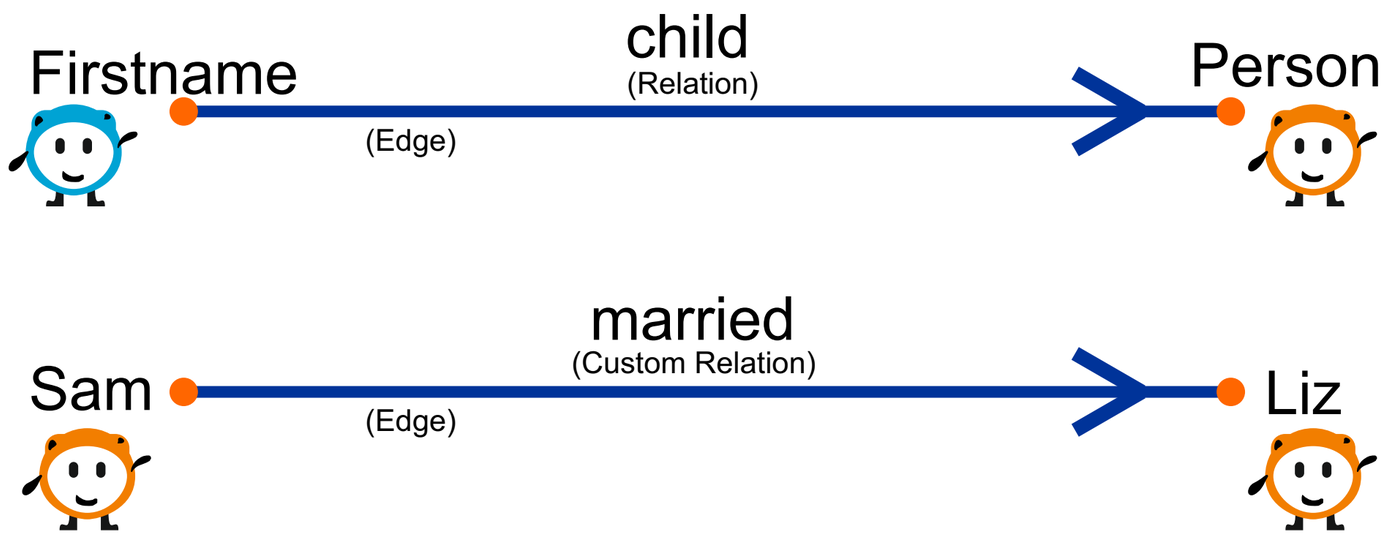
for example: "ContactForm" is a context with bots Name, Email, Phone, Message
All these bots will have "ContactForm" context object stored their contextual memory space.



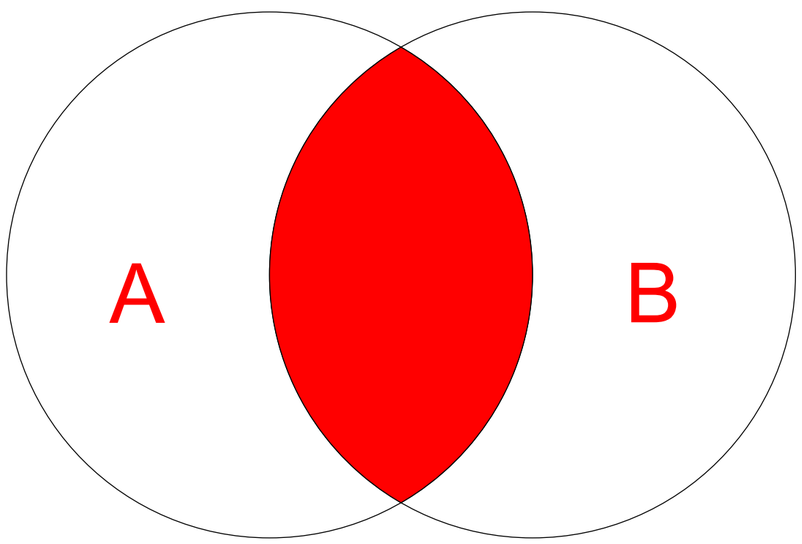




How does it work?



Example Case

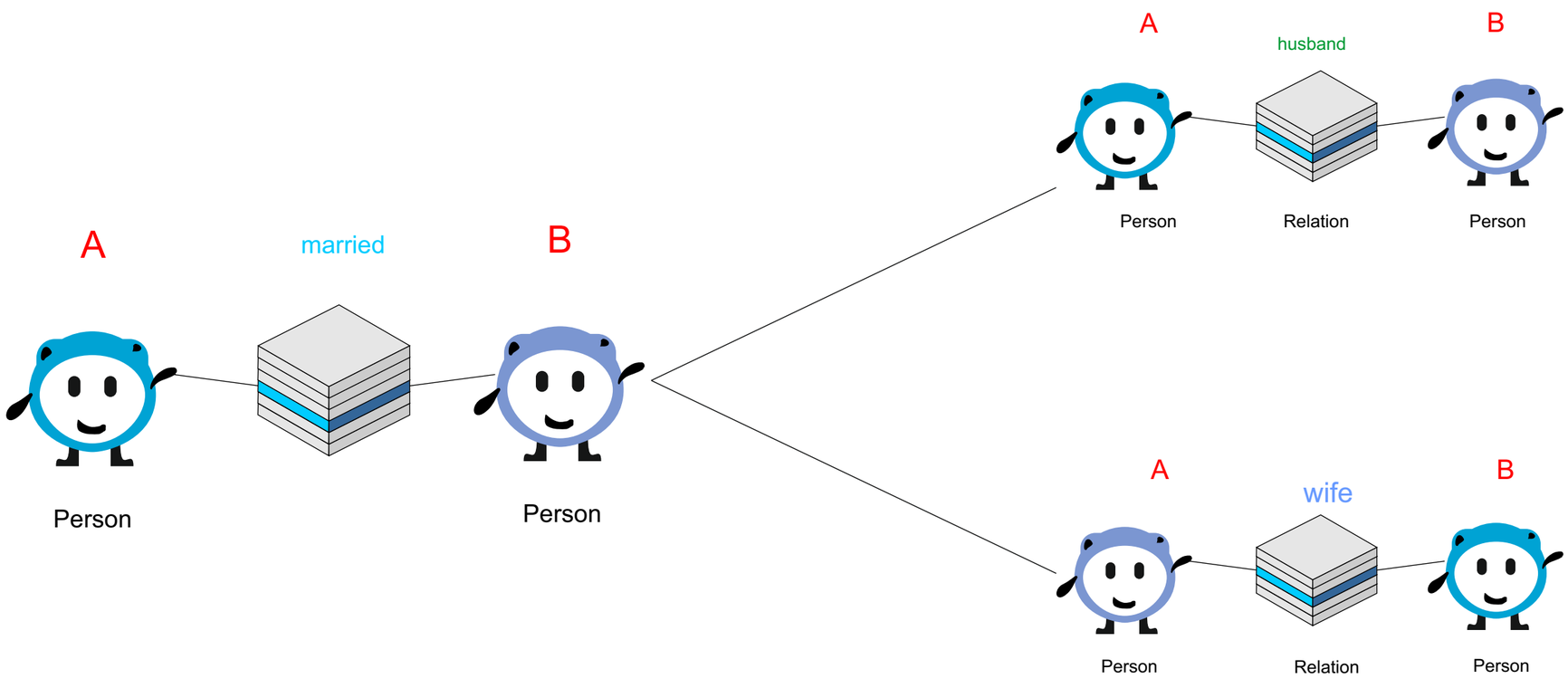


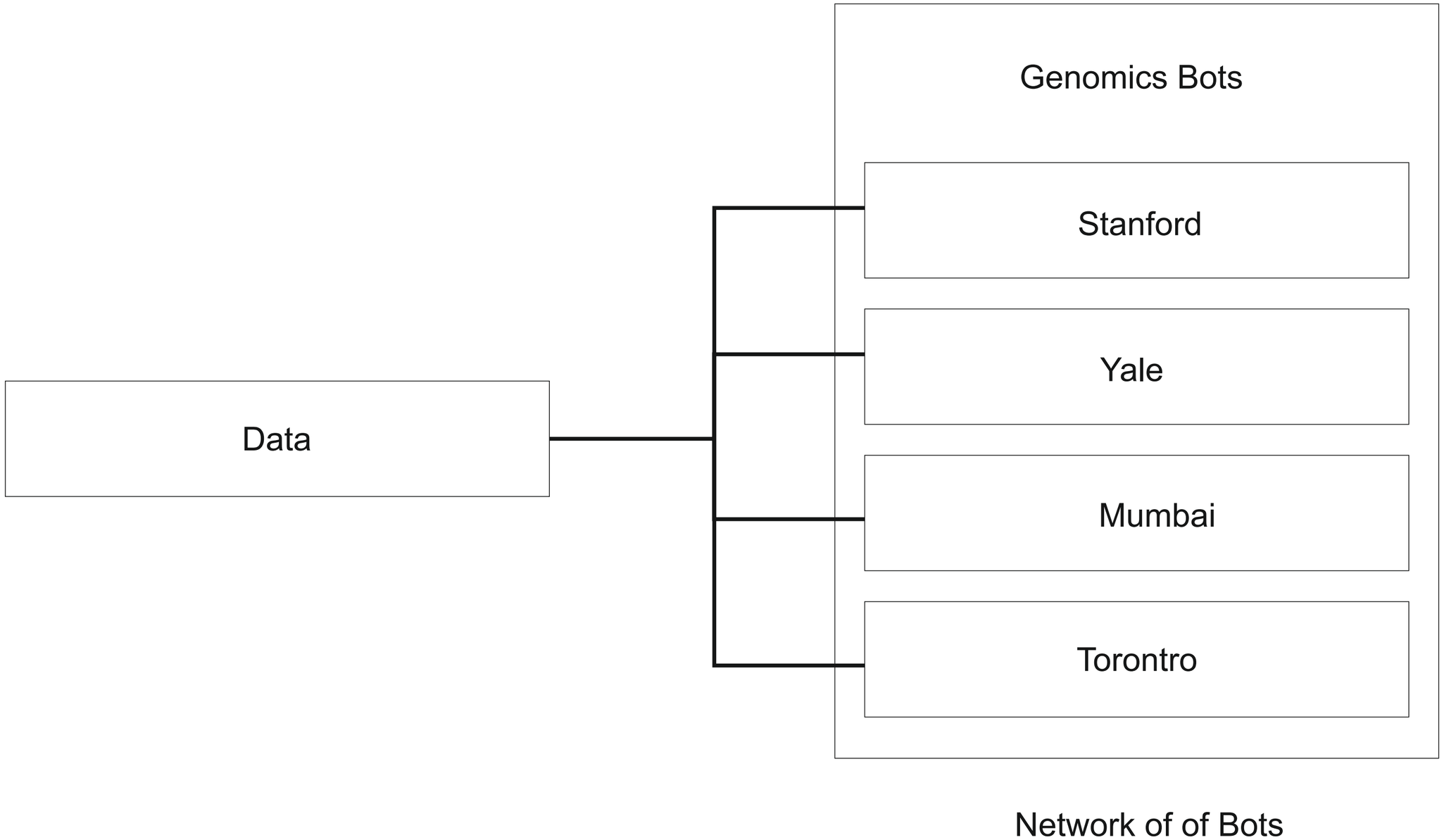
Relational memory uses structures for semantic queries with bots, edges and properties to represent relations.
Relational model among bots can be highly connected and can achieve highly complex network structure in conjunctions with other memory awareness.

Relational awareness is fully supervised memory function and it is the largest contributor in relevancy factors.

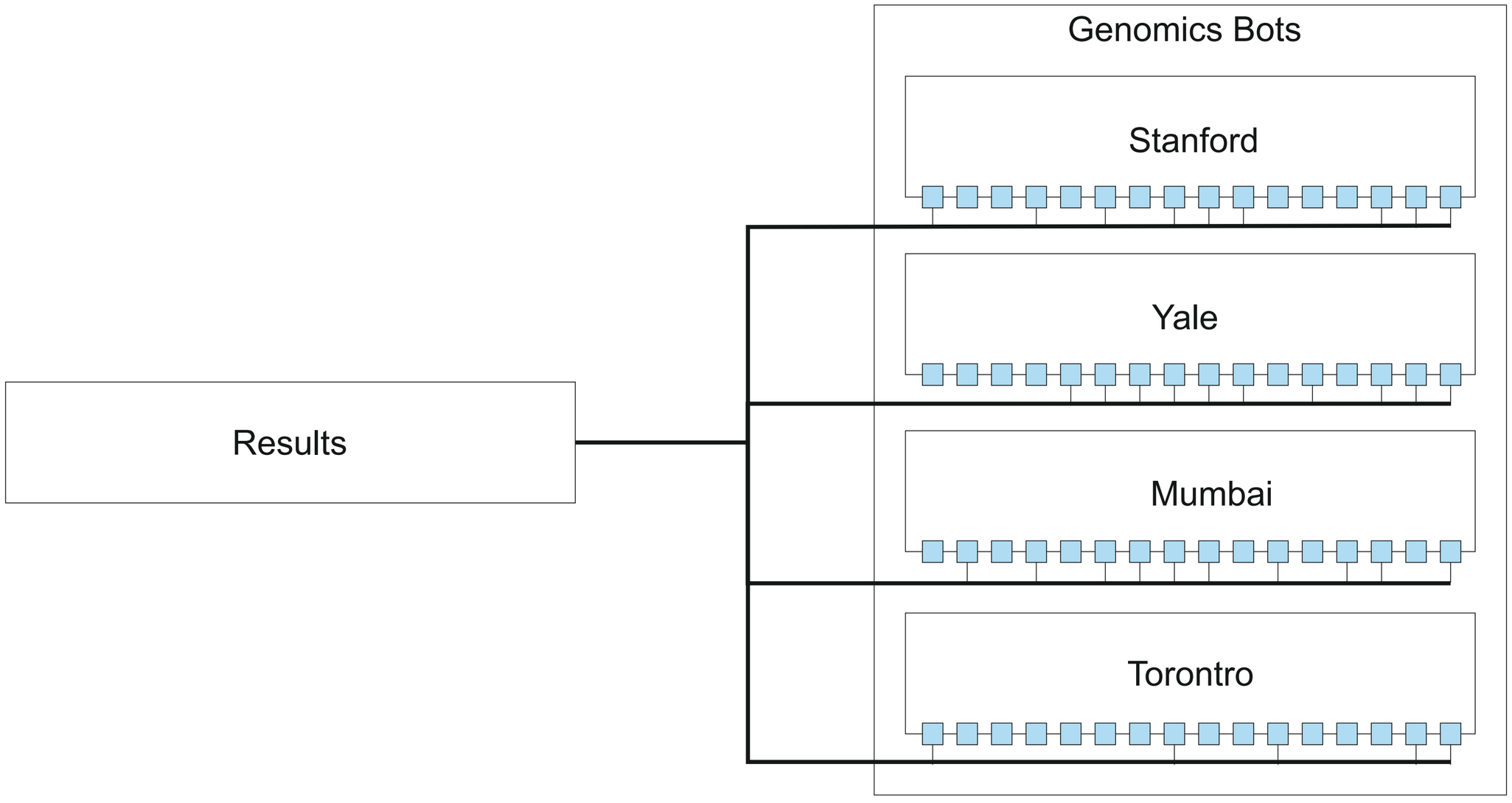
Relational can also be customized or completely custom type of awareness can be added through custom relation types.

Relation among bots can represent the data relations in database too. However It is not required to store relations in the database layer.





Network of of Bots



	BOT FRAMEWORK	MICROSERVICE ARCHITECTURE	SERVERLESS FRAMEWORK
Architecture	Smart Bot Architecture - SBA	Service Oriented Architecture - SOA	Function as a Service - FaaS
Platform	Platform Agnostic	Platform Agnostic	Platform Agnostic
System	Distributed system	Distributed system	Distributed system
License	Open source	Open source	Open source
State Mgmt	Stateful OR Stateless	Almost always stateless	Always Stateless
Intelligence	Memory and Awareness technology	No	No
Operation Cost	Reduced Operation Cost	Reduced Operation Cost	Reduced Operation Cost
Computation	Initial Computation High, subsequent low compute	Medium Computation Cost	Low Computation Cost
Development Cost	Negligible to Zero	Reduced Development Cost	Reduced Development Cost
Horizontal Scaling	Automatic	Semi-automatic	Automatic
Deployment complexity	Significantly low	Reduced	Reduced
Packaging complexity	Significantly low	Reduced	Reduced
System Administration	Not requred	Requires System Administration	Not required
Architecture Focus	Focus on Bot (can supports function/service/others)	Focus on Services with One Task in Mind	Focus on Functions
Capability	Built around bots purpose (Orders, Wishlist, Sellers, Products)	Built around individual business capability (Orders, Wishlist, Sellers, Products)	Built around functions(Save, Load, Sort)
Channels	Supports synchronous or asynchronous channels	Supports synchronous or asynchronous channels	-
Hardware Vendor	Flexible, No vendor Lock-in	Flexible, No vendor Lock-in	Vendor Lock-in (AWS Lamda, Google Cloud Functions, Azure Functions)
Execution limits	Long-lived tasks permitted	Long-lived tasks permitted	Long-lived tasks are not suited due to execution duration limits
Latency	Reduced Latency Problems	Reduced Latency Problems	Multi-Latency Problems (Cold Start)
Security	Low Security Concern (Self secure bots)	Medium Security Concern	High Security Concern
Communication Protocol	Uses lightweight Messaging	Uses lightweight Messaging	Uses lightweight Messaging
Development	Loosely coupled - Swap out bots without rebuilding	Loosely coupled - Swap out without rebuilding	Loosely coupled - Swap out without rebuilding
Entry and Exit Point	Dynamic	Well-Defined	Well-Defined Entry Point and Exit Point
Data/Datamodels	Every Bot handles its own datamodels, Databots handle data	Every microservice handles its own datamodel and data	Independent Logics and datamodels
Resource provisioning	Automatic	Requires resource provisioning	Automatic resource provisioning
Style	Suited for any style (event, service, purpose, intelligence)	Suited for service-driven style	Suited for event-driven style
Development	Intutive and faster	Easy and Fast	Easy and Fast

Monolithic Architecture has all the functions within single process and scales by replicating entire process onto multiple servers. Microservice architecture separates each functionality into services which allows scaling of individual services across servers.

Bot Framework Architecture is much like microservices, seperates each functionality as purposeful bots. These bots then scale across servers for their purpose. Bots are generally organized around business capabilities like microservices however bots are very flexible and abstraction is not limited in anyway.

Bot Framework Architecture operates on decentralized governance meaning there's no specific or standardised technology platforms. The framework can be extended to any platform as long as the structure is compatible.

There is a natural correlation between metabots and contextual boundaries that provides abstractions. They are automatic based on memory/awareness however also include reinforce business capabilities and separations. Bots handle responsibility of decentralizing data across which has implications for managing updates.

BOT FRAMEWORK : MVP

Build CLI for bot framework with operation commands

Commands for basic operations

Create bot <<botname>>
Create a Metabot in the bots directory. Handle duplicate exception.
Secondary input requests "alias", "license/ownership" and memory parameters
By default all new bots created are metabot unless type is provided.

Create databot <<botname>>
Create a Databot in the bots directory. Handle duplicate exception.
Secondary input requests "data type: text, number, date, image etc"
Next input requests "alias", "license/ownership" and memory parameters

<<command>> bot <<botname>>
Other commands are Clone, Delete, List, Disable, Enable...

Commands for memory operations

Reset bot <<botname>>
Resets bot awareness to its default state and flavour

Analyse bot <<botname>>
Quick analysis on bot's awareness, returns connected bot in relevancy factors range

Build bot <<botname>> or Rebuild bot <<botname>>
Calculates and rebuilds the relevancy factors among connected bots

Optimize bot <<botname>>
Optimises frequency channels among connected bots
Secondary input "contextual", "temporal"
optional params "rf range", "review", "clean"

Commands for remote operations

Fork bot <<usernodebot>>.<<botname>>
Forks bot from central directory of bots on the cloud.

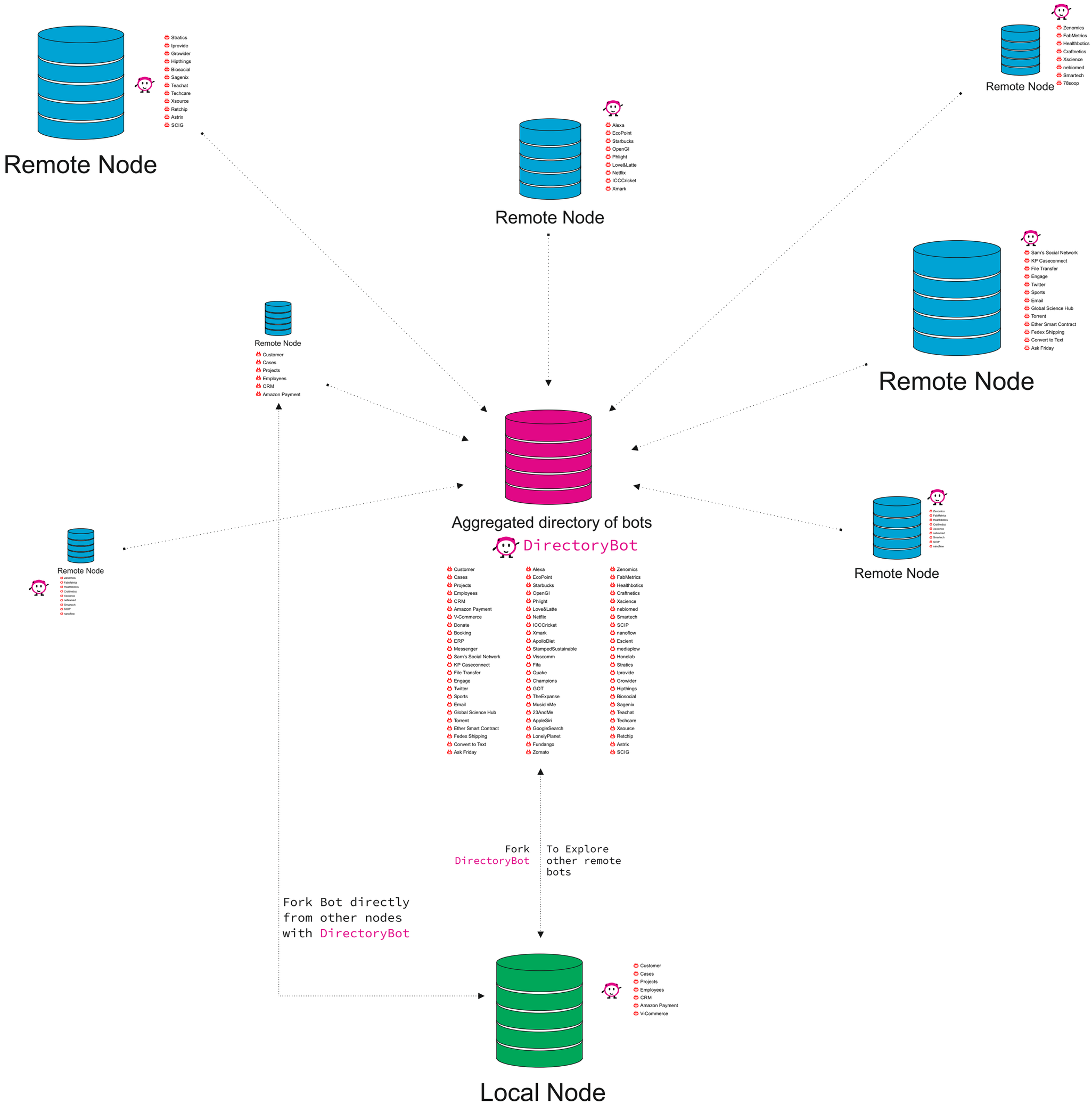
Publish bot <<usernodebot>>.<<botname>>
Adds bot to central directory on the cloud.

Update bot <<usernodebot>>.<<botname>>
Updates new rivision to the bot on central directory.

Rundown of command operations

```
#create databot command. databot will ask input type.

>>create databot firstname
    define input type.
>>text
    databot "firstname" is created

>>create databot lastname
    define input type.
>>text
    databot "firstname" is created


>>create databot dob
    define input type.
>>date
    databot "dob" is created


>>create databot phone
    define input type.
>>number
    databot "phone" is created

>>create databot email
    define input type.
>>text
    databot "email" is created

#create metabot command.

>>create metabot person
    metabot "person" is created

>>list databots
    [firstname], [lastname], [dob], [phone], [email]

>>list bots
    [firstname], [lastname], [dob], [phone], [email], [person]



#Defining relations.
{
    >>define relation person.has    #1
        new relation "has" added to person.
    >>add firstname person.has      #2
    >>add lastname person.has       #3
    >>add dob person.has            #4
    >>add phone person.has          #5
    >>add email person.has          #6

    #operation
    #1. This command defines a new relation, basically creates a directory called "has" inside person->awareness->relational
    #2. adds firstname bot #ref inside person->awareness->relational->has, also adds "_has" directory inside firstname->awareness->relational and adds person
metabot #ref index.
    #3 .. #6. similar as #2

    #note: everytime a relation is added between 2 entities. if primary entity relation is "has" then secondary entity will have "_has".
}


#Defining context
{
    >>define context person.basicInfo           #1
        new context basicInfo added to person.
    >>add firstname person.basicInfo            #2
    >>add lastname person.basicInfo              #3
    >>add dob person.basicInfo                   #4

    #operation
    #1. This command defines a new context, basically creates a directory called "basicInfo" inside person->awareness->contextual
    #2 #4. adds directory basicInfo to these bots contextual awareness, these bots share context and have same bot #ref index.
}

#Defining another context
{
    >>define context person.contactInfo         #1
        new context contactInfo added to person.
    >>add phone person.contactInfo              #2
    >>add email person.contactInfo              #3

    #operation
    #1. New context, contactInfo in contextual awareness of person Metabot
    #2 #3. adds directory contactInfo to these bots contextual awareness, these bots share context and have same bot #ref index.
}

#fetches all the context for person Metabot
>>list context person
    2 contexts found.
    "basicInfo", "contactInfo"

#fetches bots from context person.basicInfo
>>list bots person.basicInfo
    3 bots with same context
    firstname, lastname, dob

#fetching works same for other bots with shared context eg.
>>list bots firstname.basicInfo
    3 bots with same context
    lastname, dob, person



>>list bots person.context.*
    6 bots found with same context
    firstname, lastname, dob, phone, email


>>person>>list relations
    1 relations found with 4 bots
    has>firstname
    has>lastname
    has>dob
    has>phone

//Nested commands for bot
>>select person
>>person>>list relations
>>person>>list context
>>person>>list bots in relation.*
>>person>>list bots in relation.has
>>person>>list bots in context.*
```

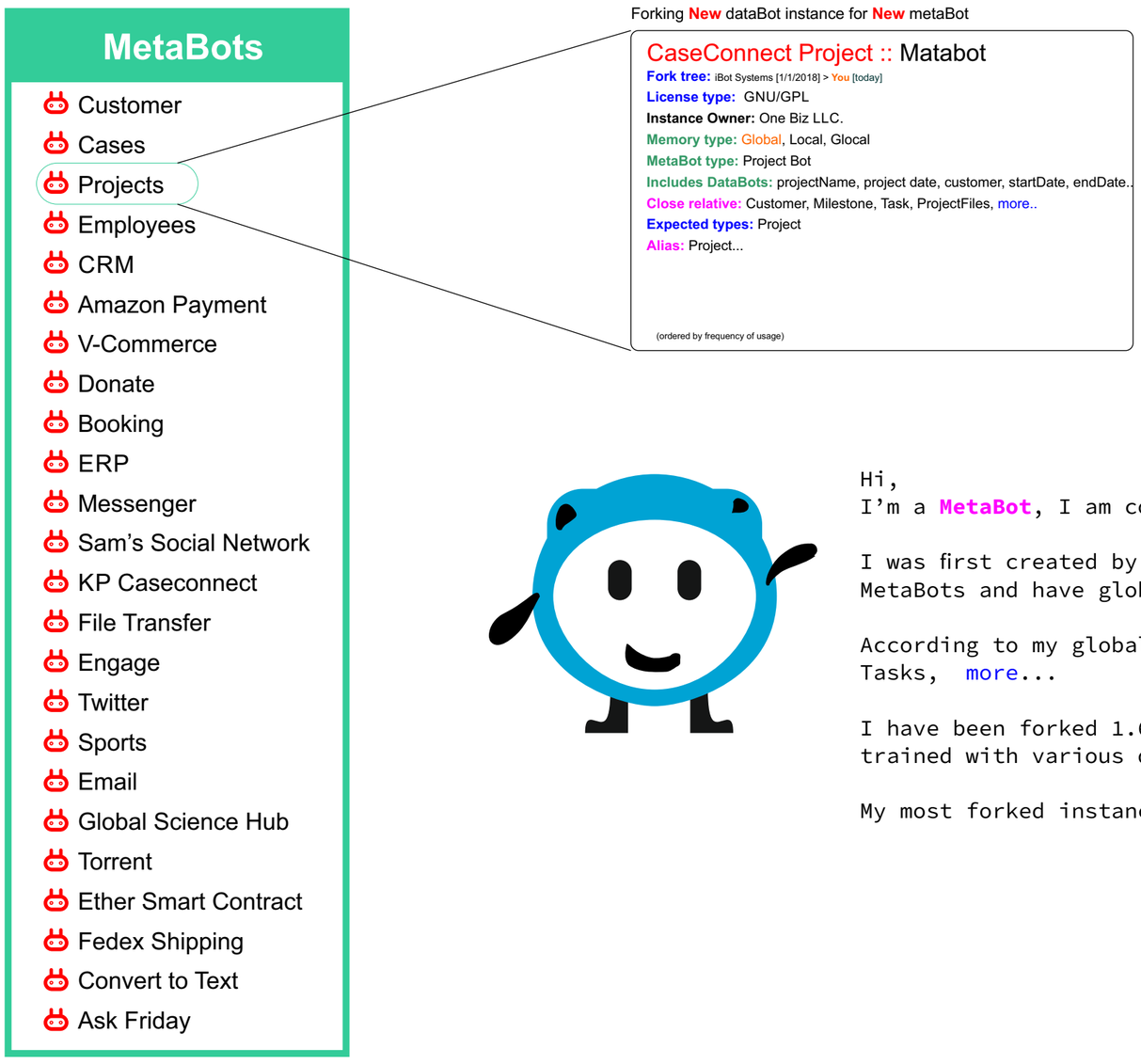
Bots Directory



Metabots

A MetaBot is a like a DataBot but it's coupled with higher level abstraction such as models, logics, process etc. it follows loose naming standards and contains utility cognition of of its scope. Metabots can also be forked from other working instances to inherit cognition. Metabots usually contain other bots to represent higher order of abstractions. Similar to Databots, it represents a bot function using memory/awareness, state and interfaces.
MetaBots also have universally shared libraries which others can fork and instantiated, its internal schema and memory will be develop separately and remain exclusive, however an experienced instance of can also be forked to inherit global/glocal, memory and intelligence from its previous owner's environment and states.

Pink is evolving type. Blue is default and fixed type. Green is interchangeable



Hi,
I'm a **MetaBot**, I am coupled with Project, also knows as ProjectBot, more...

I was first created by iBot systems on 1/1/18. I have been used in 1.5 thousand unique MetaBots and have global memory of 63Mb.

According to my global memory, my closest relative databots are Customers, ProjectFiles, Milestones, Tasks, more...

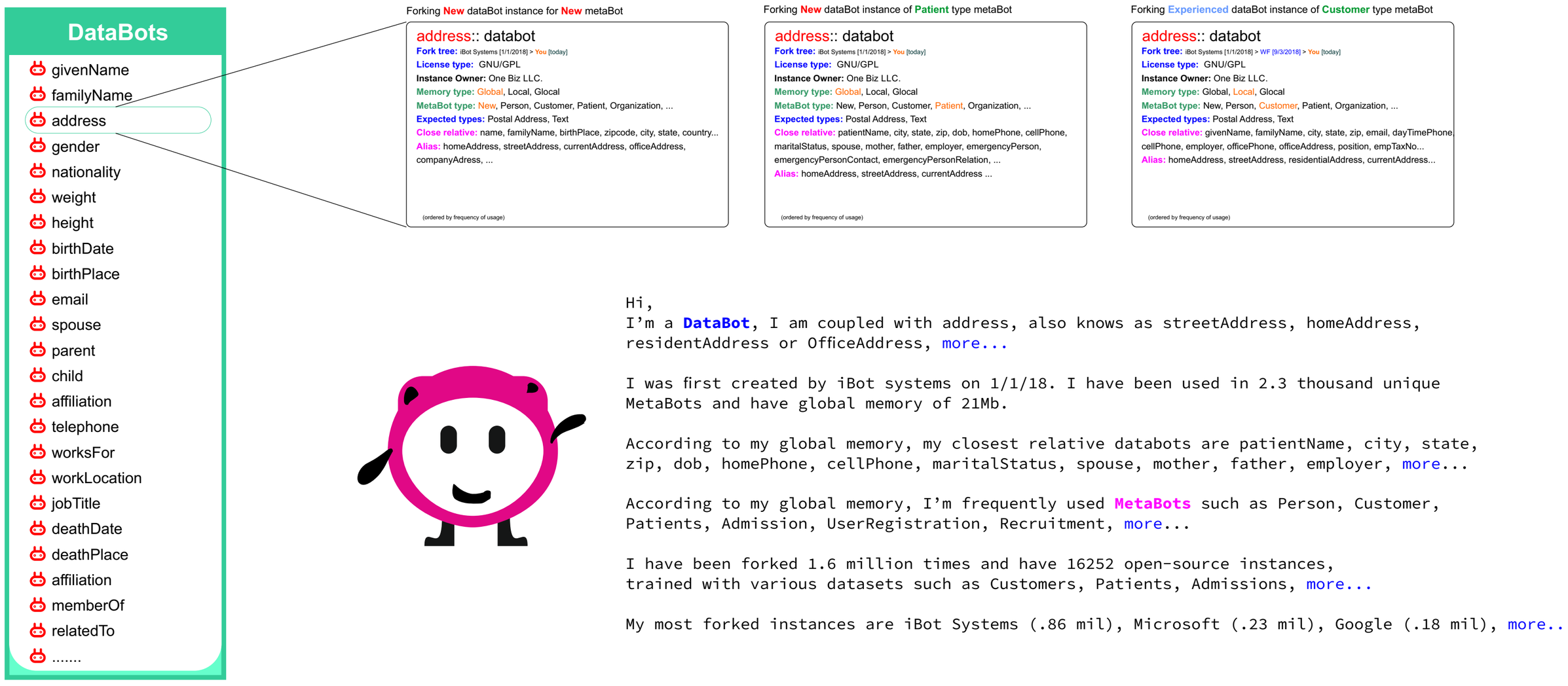
I have been forked 1.6 million times and have 16252 open-source instances, trained with various datasets such as Manufacturing, Automation, SoftwareDev, more...

My most forked instances are iBot Systems (.86 mil), Github (.23 mil), Amazon (.18 mil), more...

Databots

Databots are like variables such as string or Int but represents higher order of abstraction, coupled with single piece of information such as givenName, birthPlace, address etc. It is the layers that communicated with database layer, This single piece of information correlates with a node in database. They are universally shared libraries built upon strict naming standard. when new Databot is instantiated, its internal schema and memory will be develop separately and remain exclusive, however an experienced instance of databot can also be forked, which will inherit global/glocal, memory and intelligence from its previous owner's environment and states.

Pink is evolving type. Blue is default and fixed type. Green is interchangeable



Hi,
I'm a **DataBot**, I am coupled with address, also knows as streetAddress, homeAddress, residentAddress or OfficeAddress, more...

I was first created by iBot systems on 1/1/18. I have been used in 2.3 thousand unique MetaBots and have global memory of 21Mb.

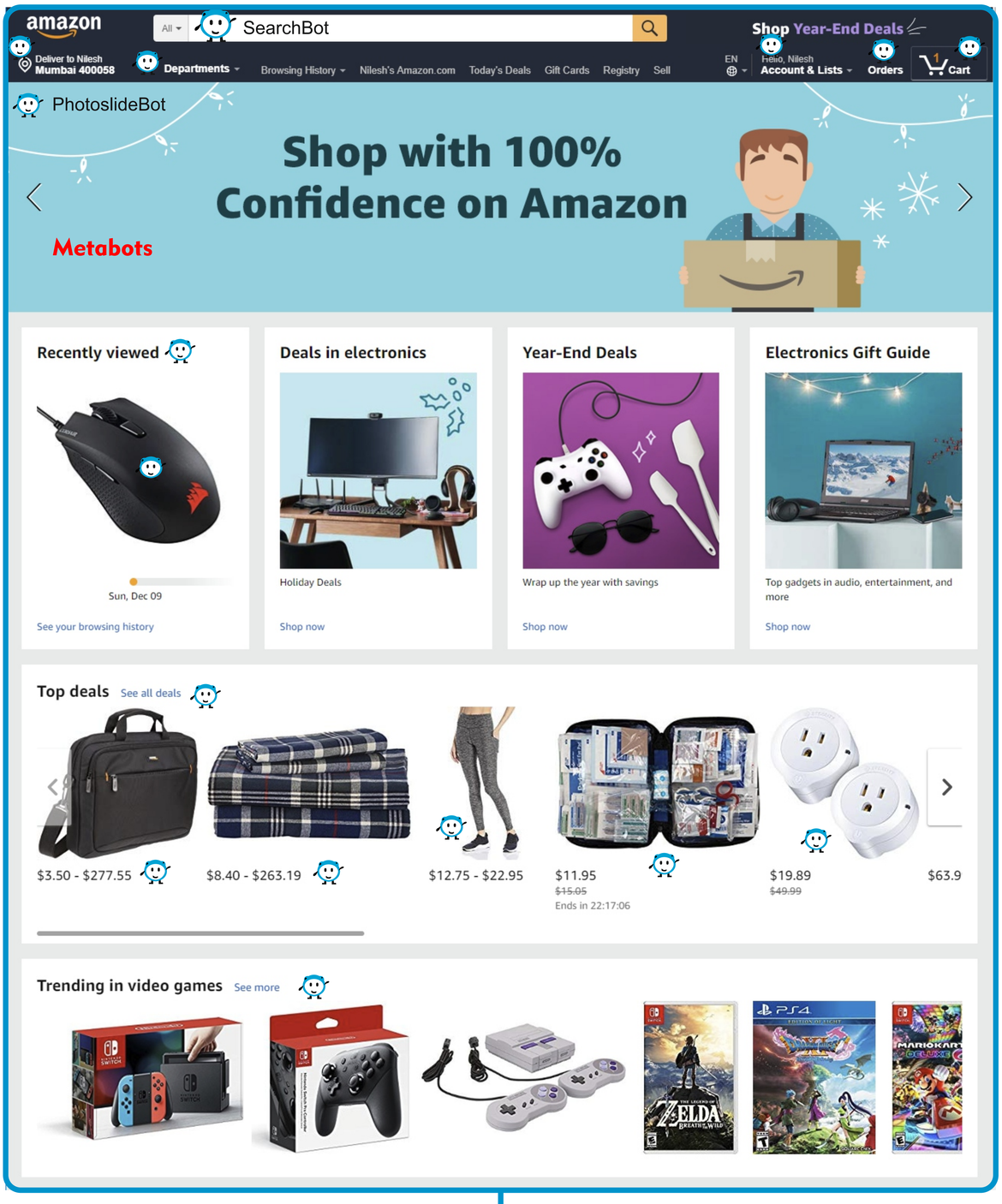
According to my global memory, my closest relative databots are patientName, city, state, zip, dob, homePhone, cellPhone, maritalStatus, spouse, mother, father, employer, more...

According to my global memory, I'm frequently used **MetaBots** such as Person, Customer, Patients, Admission, UserRegistration, Recruitment, more...

I have been forked 1.6 million times and have 16252 open-source instances, trained with various datasets such as Customers, Patients, Admissions, more...

My most forked instances are iBot Systems (.86 mil), Microsoft (.23 mil), Google (.18 mil), more...

BOTS interaction example



Metabot : Homepage flavor
AmazonUSHome

```
M:AmazonUSHome Bot
    M:LogoBot
        D:Logo - Image
        D:Width - Number
        D:Height - Number
    M:CartBot
        M:CartProduct
            M:Product
                D:ProductName - Text
                D:SKU - Number
                D:Price - Number
                ...
            D:Qty - Number
            D:...
            ...
        M:...
    M:...
```

In this example, M prefix is metabot and D prefix is databot, page is "AmazonUSHome" metabot and all bots are used within the context. Since all bots have memory and know their purpose, they have binding relevancy factors among them. This method enabled rapid development and deploying capability unlike any other framework exist today.

The relevancy factors are calculated along with usage over time based on Temporal, Contextual and Relational awarenesses. Bot framework escapes limitation of conway's law by utilizing these memory and awareness system which helps in building a specific intelligence for a bot purpose. This then can be used over and over just by recalling from bots memory based on connection (RF). This method saves huge waste of human effort for development, management, and other resources involved.

These bots can be forked and licensing can be nested, this opens up the opportunities to build logistical intelligence framework through optimized system design.

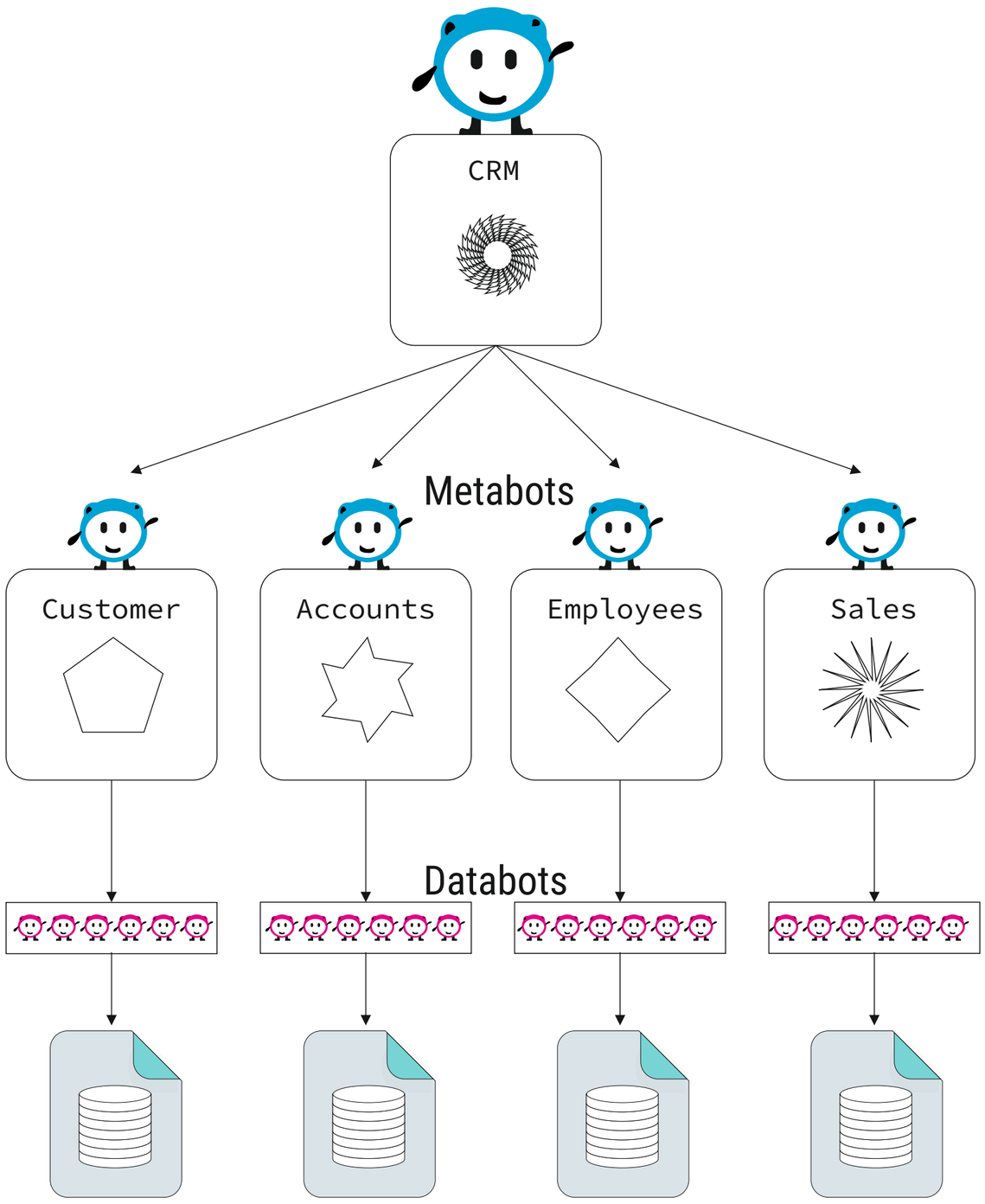
This is just an example case however same function can be applied to any business process such as Medical, Finance, Personal, Products, Services etc.

What are the techniques in Bot Framework?

Bot framework is a lightweight web container that is a strong option for development of bots much similar to microservices. Framework is asynchronous in nature while developers can build bots to add any functionality. Use of this framework depends on easier development and faster deployability. A greater use is expected by applications/bots developers.

Developers can use bot framework as a development platform considering it reduces development cost significantly while supporting all platforms. Bots can also be used as graph like databases (Neo4j) which store information as arbitrarily interconnected nodes linked relations among bots. It's possible to completely eliminate tables and joins. Since Bots architecture is memory based therefore highly adaptable and extensible, they are schema-less and an excellent choice for modeling semi-structured data in complex domains. Bot framework offers light messaging via Meta Query Language as well as Meta Scripting Language which is dynamically derived to support human readable language making it easy to learn.

Bot framework is decentralized in nature however unlike blockchains, Bots can be stored on Inter Planetary File System (IPFS), which is a content-addressed (in this case bots address), versioned, peer-to-peer file system. It’s designed to distribute high volumes of data with high efficiency and removed from any centralized authority. Bots are stored on peers that don’t need to trust each other. IPFS keeps every version of a file so you never lose important files. Bots+IPFS is a great combination in blockchains.



Each bots have independent database

Blueprint editor enables graphical user interfaces and flows in order to create applications. It's not like low-code platforms but automation platform. Low-code platforms lack support of complex development. Bot framework allows benefit of low code with automation while still maintaining the ability to write complex codes to create high-quality software. This will bridge the gap between development platforms to include non-technical people who can build higher abstraction logics.

Bot framework is fundamentally a collaborative development environment that lets anyone easily fork (copy) and repurpose existing bots or create your own bots. Basic sentiment is that people who have good intuition of business logics can rapidly build and test the bots, also makes it ideal for people wanting to learn. Developers can build bots and add special purpose awareness which then can be licensed by others. This adds to motivation to do collaborative programming or building bots.

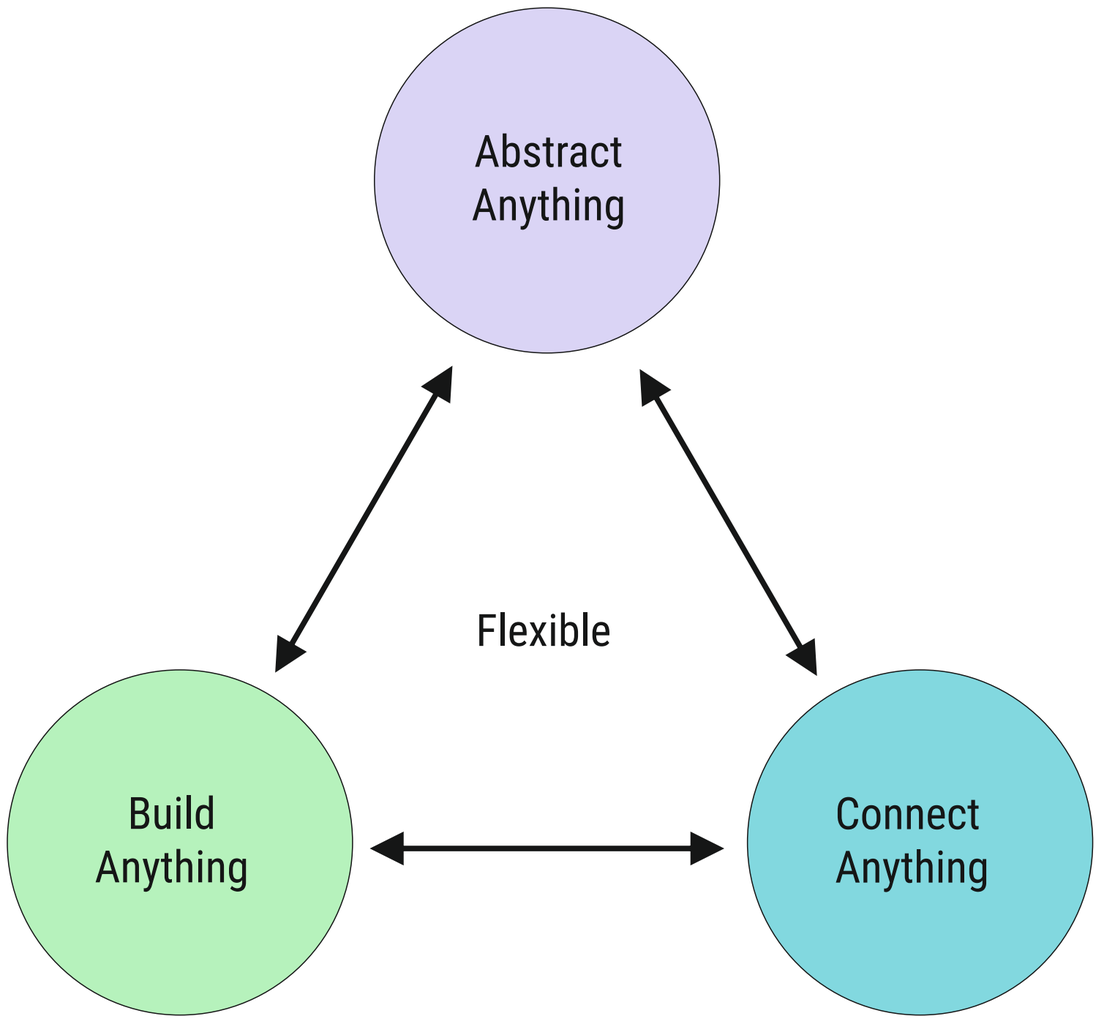
Bots provides capabilities to model types for metadata, add layers of abstractions, track the data lineage and enable data discovery. This is dynamic and adpatable since group of bots can be assigned to perform specific function which then classify assets in the form of bots. This falls under approach of evolutionary database design practices.

What can you do with Bot Framework?

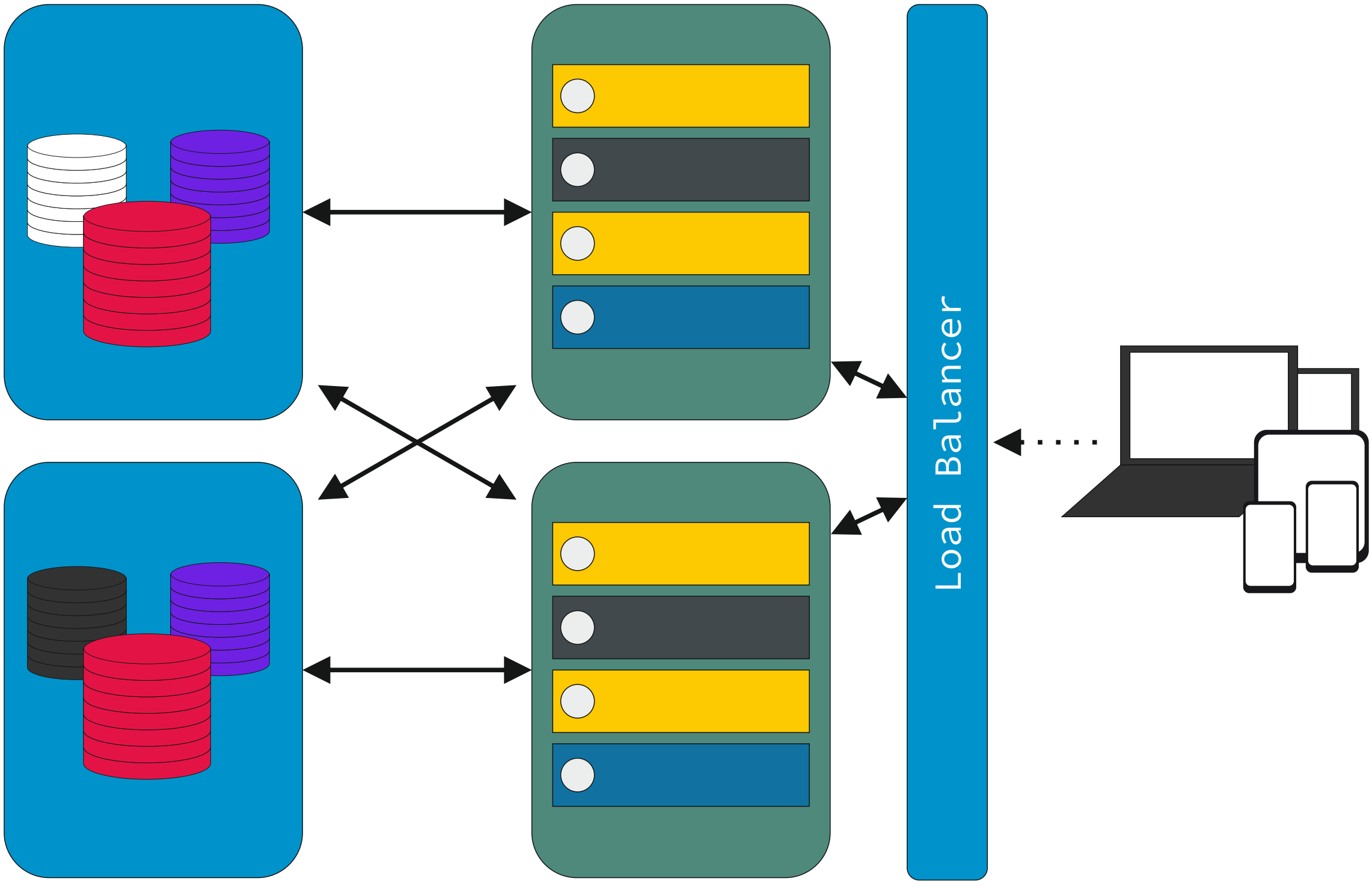
Bot framework is imagined to get you up and running as quickly as possible, with minimal upfront development and confuguration. Framework assigns higher abstraction perspectives to building smart applications or bots.

Bots are directly created in the online interface as innovative approach to building enterprise ready systems. It simplifies distributed architecture by implementing already proven technoloy such as microservices.

Connect to anything or Internet of Anything, such as mobile devices, sensors, wearables, automobiles, and more. Bot Framework provides a unified service for creating bots that address streaming and ETL-based data processing patterns.



Scalability of Monolithic Architecture



Scalability with Bot Framework